VB NET EJERCICIOS

300 ejercicios FORM de principiante a avánzado

Luis Minguillón Pascual

VB NET EJERCICIOS

300 ejercicios de principiante a avánzado

Luis Minguillón Pascual

COPYRIGHT

La licencia de este libro electrónico es para uso personal. Por lo tanto no puedes revenderlo a otras personas.

Gracias por respetar los derechos de autor.

©Luis Minguillón Pascual
13/09/2025

Prefacio

Este libro nace con la intención de ser una guía práctica y accesible para todas aquellas personas que deseen aprender a programar en **Visual Basic .NET con formularios (Windows Forms)**. He reunido en estas páginas un total de **300 ejercicios**, organizados desde el nivel principiante hasta un nivel avanzado, buscando siempre que la explicación de cada uno sea clara, comprensible y útil.

La idea que me ha movido a crear esta serie de libros es sencilla pero profunda: **el conocimiento no debe quedarse guardado, debe compartirse**. Porque el conocimiento nos abre puertas, nos hace libres y nos permite crecer como individuos y como sociedad.

En cada ejercicio he puesto la mirada en la sencillez y la claridad, con el deseo de que cualquiera pueda entenderlos sin importar su punto de partida. La programación, más que un conjunto de reglas, es un lenguaje para expresar ideas y resolver problemas, y mi propósito ha sido acercar ese lenguaje a todos.

No puedo dejar de mencionar a **mi mujer, Marisa**, quien ha estado a mi lado en todo momento. Su apoyo incondicional ha sido fundamental, y sin él, esta serie de libros simplemente no existiría.

Espero de corazón que este trabajo no solo te ayude a aprender, sino que también te inspire a compartir lo que aprendas con otros. Porque el saber cobra su verdadero sentido cuando se transmite.

Luis Minguillón Pascual

Índice

1. **Formulario con un botón que muestra un mensaje**
2. **Formulario con una etiqueta y un botón que cambia el texto**
3. **Formulario con un cuadro de texto y un botón para mostrar su contenido**
4. **Formulario con dos botones que cambian el color de fondo**
5. **Formulario con un cuadro de texto que convierte a mayúsculas**
6. **Formulario con dos cuadros de texto para sumar números**
7. **Formulario con un botón para limpiar cuadros de texto**
8. **Formulario con un botón que cierra la aplicación**
9. **Formulario con una etiqueta que muestra la hora actual**
10. **Formulario con un cuadro de texto que cuenta caracteres**
11. **Formulario con dos botones para mostrar imágenes diferentes**
12. **Formulario con un botón que cambia el tamaño de la ventana**
13. **Formulario con un cuadro de texto que solo permite números**
14. **Formulario con tres botones que cambian el color de texto**
15. **Formulario con un cuadro de texto y un botón para invertir texto**
16. **Formulario con un contador de clics**
17. **Formulario con dos botones para mostrar mensajes distintos**
18. **Formulario con un CheckBox que habilita o deshabilita un botón**
19. **Formulario con un TextBox multilinea y un botón para contar líneas**
20. **Formulario con una etiqueta que cambia de color al pasar el mouse**
21. **Formulario con un botón que cambia el ícono del formulario**
22. **Formulario con un TrackBar que controla el tamaño del texto**
23. **Formulario con un cuadro de texto que valida correo electrónico**
24. **Formulario con un botón que rota imágenes**
25. **Formulario con una lista de opciones en un ComboBox**
26. **Formulario con un ListBox que agrega elementos desde un TextBox**
27. **Formulario con un botón que elimina el elemento seleccionado del ListBox**
28. **Formulario con un cuadro de texto que convierte a número romano**
29. **Formulario con un botón que genera un número aleatorio**
30. **Formulario con un temporizador que actualiza la hora**
31. **Formulario con un botón que oculta y muestra una etiqueta**
32. **Formulario con un botón que abre un nuevo formulario vacío**
33. **Formulario con una barra de progreso que avanza al presionar un botón**
34. **Formulario con un botón que copia texto al portapapeles**
35. **Formulario con un cuadro de texto que limita caracteres**
36. **Formulario con un botón que abre un cuadro de diálogo de color**
37. **Formulario con un botón que abre un cuadro de diálogo de fuente**
38. **Formulario con un botón que abre un cuadro de diálogo de archivo**
39. **Formulario con un botón que abre un cuadro de diálogo de carpeta**
40. **Formulario con un menú simple con dos opciones**
41. **Formulario con un menú que cambia el color de fondo**
42. **Formulario con un menú que muestra un mensaje**
43. **Formulario con un menú para abrir otro formulario**

44. **Formulario con una barra de herramientas (ToolStrip)**
45. **Formulario con un cuadro de texto que busca en un ListBox**
46. **Formulario con un botón que cambia el título del formulario**
47. **Formulario con dos botones que mueven un Label en el formulario**
48. **Formulario con un botón que muestra la fecha actual**
49. **Formulario con un PictureBox que muestra diferentes imágenes**
50. **Formulario con un botón que guarda texto en un archivo**
51. **Agenda de contactos básica con ListBox**
52. **Calculadora simple (suma, resta, multiplicación, división)**
53. **Conversor de temperatura (Celsius-Fahrenheit)**
54. **Conversor de divisas**
55. **Bloc de notas básico**
56. **Aplicación que guarda y carga texto desde archivo**
57. **Aplicación que cuenta palabras en un TextBox**
58. **Aplicación que muestra tablas de multiplicar**
59. **Aplicación que genera contraseñas aleatorias**
60. **Aplicación que muestra mensajes aleatorios**
61. **Aplicación que calcula el IMC**
62. **Conversor de unidades de longitud (metros a cm, mm, km)**
63. **Conversor de unidades de peso (kg a g, lb)**
64. **Conversor de unidades de tiempo (horas a minutos y segundos)**
65. **Conversor de unidades de velocidad (km/h a m/s)**
66. **Aplicación que guarda configuraciones en My.Settings**
67. **Calculadora de área de figuras geométricas (círculo y cuadrado)**
68. **Calculadora de perímetro de figuras geométricas (círculo y cuadrado)**
69. **Juego del número secreto**
70. **Juego de adivinar palabra**
71. **Juego de piedra, papel o tijera**
72. **Cronómetro con Start, Stop y Reset**
73. **Temporizador con alarma**
74. **Editor de listas (agregar, modificar y borrar)**
75. **Editor de texto con guardar y abrir archivo**
76. **Aplicación con varios formularios enlazados**
77. **Agenda que guarda datos en archivo de texto**
78. **Agenda que guarda datos en archivo CSV**
79. **Agenda que guarda datos en archivo XML**
80. **Agenda que guarda datos en base de datos Access**
81. **Formulario con gráficos simples**
82. **Formulario con impresión de texto**
83. **Calculadora de ecuaciones de segundo grado**
84. **Calculadora de interés simple**
85. **Calculadora de interés compuesto**
86. **Conversor de números decimales a binarios**
87. **Conversor de binarios a decimales**
88. **Conversor de números romanos a enteros**
89. **Conversor de horas a minutos y segundos**

90. **Conversor de kilogramos a libras**
91. **Conversor de metros a pies**
92. **Conversor de Celsius a Kelvin**
93. **Conversor de binario a hexadecimal**
94. **Conversor de hexadecimal a decimal**
95. **Juego de memoria con botones**
96. **Juego de laberinto simple**
97. **Juego de dados con imágenes**
98. **Juego de ahorcado**
99. **Juego de tic-tac-toe**
100. **Juego de cartas simple**
101. **Juego de lotería con números aleatorios**
102. **Juego de ruleta simple**
103. **Juego de preguntas y respuestas**
104. **Juego de carreras con imágenes**
105. **Juego de disparos con botones**
106. **Juego de laberinto con teclas de dirección**
107. **Agenda con búsqueda por nombre**
108. **Agenda con edición de contactos**
109. **Agenda con borrado de contactos**
110. **Agenda con exportación a archivo de texto**
111. **Agenda con importación desde archivo de texto**
112. **Agenda con exportación a Excel**
113. **Agenda con importación desde Excel**
114. **Agenda con base de datos SQLite**
115. **Agenda con base de datos SQL Server**
116. **Agenda con base de datos MySQL**
117. **Agenda con base de datos PostgreSQL**
118. **Aplicación de notas con recordatorio**
119. **Aplicación de calendario simple**
120. **Agenda con calendario integrado**
121. **Agenda con alarma**
122. **Agenda con notificaciones**
123. **Agenda con múltiples formularios**
124. **Agenda con login de usuario**
125. **Agenda con niveles de usuario**
126. **Agenda con encriptación de datos**
127. **Agenda con desencriptación de datos**
128. **Calculadora científica simple**
129. **Calculadora con funciones trigonométricas**
130. **Calculadora con funciones logarítmicas**
131. **Calculadora con potencias y raíces**
132. **Calculadora con historial de operaciones**
133. **Calculadora con memoria de resultados**
134. **Juego de preguntas con puntuación**
135. **Juego de preguntas con tiempo límite**

136. **Juego de preguntas con niveles**
137. **Juego de preguntas con base de datos**
138. **Juego de preguntas con imágenes**
139. **Juego de preguntas con sonidos**
140. **Juego de preguntas multijugador**
141. **Juego de preguntas con ranking**
142. **Juego de preguntas con exportación de resultados**
143. **Juego de preguntas con importación de resultados**
144. **Juego de preguntas con estadísticas**
145. **Juego de preguntas con gráficos**
146. **Juego de preguntas con personalización**
147. **Juego de preguntas con guardado automático**
148. **Juego de preguntas con recuperación de partida**
149. **Juego de preguntas con dificultad progresiva**
150. **Juego de preguntas con logros desbloqueables**
151. **Editor de imágenes simple**
152. **Editor de audio simple**
153. **Reproductor de música básico**
154. **Reproductor de video básico**
155. **Reproductor multimedia integrado**
156. **Aplicación que controla volumen del sistema**
157. **Aplicación que controla brillo de pantalla**
158. **Aplicación que toma capturas de pantalla**
159. **Aplicación que graba sonido**
160. **Aplicación que graba video**
161. **Aplicación que reproduce listas de reproducción**
162. **Aplicación que descarga archivos de internet**
163. **Aplicación que sube archivos a servidor FTP**
164. **Aplicación que se conecta a una API REST**
165. **Aplicación que se conecta a una API SOAP**
166. **Aplicación que consume servicios web JSON**
167. **Aplicación que consume servicios web XML**
168. **Cliente de chat local simple**
169. **Cliente de chat en red local**
170. **Cliente de chat con base de datos**
171. **Cliente de chat con login de usuario**
172. **Cliente de chat con historial de mensajes**
173. **Cliente de chat con emoticonos**
174. **Cliente de chat con archivos adjuntos**
175. **Cliente de chat con múltiples salas**
176. **Cliente de chat con videollamadas**
177. **Cliente de chat con llamadas de voz**
178. **Cliente de chat con notificaciones**
179. **Cliente de chat con moderadores**
180. **Cliente de chat con administración**
181. **Cliente de chat con cifrado de mensajes**

182. **Cliente de chat con recuperación de historial**
183. **Cliente de chat con estadísticas**
184. **Cliente de chat con exportación de mensajes**
185. **Cliente de chat con importación de mensajes**
186. **Cliente de correo electrónico básico**
187. **Cliente de correo con envío SMTP**
188. **Cliente de correo con recepción POP3**
189. **Cliente de correo con conexión IMAP**
190. **Cliente de correo con adjuntos**
191. **Cliente de correo con múltiples cuentas**
192. **Cliente de correo con filtros de mensajes**
193. **Cliente de correo con carpetas personalizadas**
194. **Cliente de correo con notificaciones**
195. **Cliente de correo con contactos**
196. **Cliente de correo con calendario**
197. **Cliente de correo con recordatorios**
198. **Cliente de correo con sincronización**
199. **Cliente de correo con cifrado**
200. **Cliente de correo con firmas personalizadas**
201. **Sistema de login con logs de actividad**
202. **Sistema de login con auditoría**
203. **Sistema de login con autenticación externa**
204. **Sistema de login con validación de IP**
205. **Sistema de login con tiempo de expiración**
206. **Sistema de login con recordatorio de sesión**
207. **Sistema de login con restricción de horarios**
208. **Sistema de login con acceso remoto**
209. **Sistema de login con sincronización de datos**
210. **Sistema de login con cifrado de base de datos**
211. **Sistema de login con hash de contraseñas**
212. **Sistema de inventario básico**
213. **Sistema de inventario con base de datos**
214. **Sistema de inventario con facturación**
215. **Sistema de inventario con reportes**
216. **Sistema de inventario con gráficos**
217. **Sistema de inventario con exportación Excel**
218. **Sistema de inventario con importación Excel**
219. **Sistema de inventario con códigos de barra**
220. **Sistema de inventario con múltiples usuarios**
221. **Sistema de inventario con permisos**
222. **Sistema de inventario con auditoría**
223. **Sistema de inventario con estadísticas**
224. **Sistema de inventario con gráficos dinámicos**
225. **Sistema de inventario con múltiples almacenes**
226. **Sistema de inventario con conexión remota**
227. **Sistema de inventario con backup automático**

228. **Sistema de inventario con restauración de datos**
229. **Sistema de inventario con notificaciones**
230. **Sistema de inventario con control de lotes**
231. **Sistema de inventario con caducidades**
232. **Sistema de ventas básico**
233. **Sistema de ventas con base de datos**
234. **Sistema de ventas con clientes**
235. **Sistema de ventas con facturación**
236. **Sistema de ventas con reportes**
237. **Sistema de ventas con estadísticas**
238. **Sistema de ventas con gráficos**
239. **Sistema de ventas con exportación Excel**
240. **Sistema de ventas con importación Excel**
241. **Sistema de ventas con múltiples usuarios**
242. **Sistema de ventas con permisos**
243. **Sistema de ventas con auditoría**
244. **Sistema de ventas con conexión remota**
245. **Sistema de ventas con backup automático**
246. **Sistema de ventas con restauración de datos**
247. **Sistema de ventas con notificaciones**
248. **Sistema de ventas con control de stock**
249. **Sistema de ventas con caducidades**
250. **Sistema de ventas con código de barras**
251. **Sistema de ventas con descuentos**
252. **Sistema escolar básico**
253. **Sistema escolar con base de datos**
254. **Sistema escolar con registro de alumnos**
255. **Sistema escolar con registro de profesores**
256. **Sistema escolar con registro de materias**
257. **Sistema escolar con registro de calificaciones**
258. **Sistema escolar con reportes**
259. **Sistema escolar con gráficos**
260. **Sistema escolar con exportación Excel**
261. **Sistema escolar con importación Excel**
262. **Sistema escolar con múltiples usuarios**
263. **Sistema escolar con permisos**
264. **Sistema escolar con auditoría**
265. **Sistema escolar con conexión remota**
266. **Sistema escolar con backup automático**
267. **Sistema escolar con restauración de datos**
268. **Sistema escolar con notificaciones**
269. **Sistema escolar con estadísticas**
270. **Sistema escolar con boletines**
271. **Sistema escolar con login de usuarios**
272. **Sistema médico básico**
273. **Sistema médico con registro de pacientes**

274. **Sistema médico con registro de doctores**
275. **Sistema médico con registro de citas**
276. **Sistema médico con historial clínico**
277. **Sistema médico con facturación**
278. **Sistema médico con reportes**
279. **Sistema médico con gráficos**
280. **Sistema médico con exportación Excel**
281. **Sistema médico con importación Excel**
282. **Sistema médico con múltiples usuarios**
283. **Sistema médico con permisos**
284. **Sistema médico con auditoría**
285. **Sistema médico con conexión remota**
286. **Sistema médico con backup automático**
287. **Sistema médico con restauración de datos**
288. **Sistema médico con notificaciones**
289. **Sistema médico con recordatorios de citas**
290. **Sistema médico con estadísticas**
291. **Sistema médico con login de usuarios**
292. **Aplicación que toma capturas de pantalla** *(Nota: el programa 158 ya listado aquí es el mismo)*
293. **Aplicación que graba sonido** *(Nota: el programa 159 ya listado aquí es el mismo)*
294. **Aplicación que graba video** *(Nota: el programa 160 ya listado aquí es el mismo)*
295. **Aplicación que reproduce listas de reproducción** *(Nota: el programa 161 ya listado aquí es el mismo)*
296. **Aplicación que descarga archivos de internet** *(Nota: el programa 162 ya listado aquí es el mismo)*
297. **Aplicación que sube archivos a servidor FTP** *(Nota: el programa 163 ya listado aquí es el mismo)*
298. **Aplicación que se conecta a una API REST** *(Nota: el programa 164 ya listado aquí es el mismo)*
299. **Aplicación que se conecta a una API SOAP** *(Nota: el programa 165 ya listado aquí es el mismo)*
300. **Aplicación que consume servicios web JSON** *(Nota: el programa 166 ya listado aquí es el mismo)*

Programa 1. Formulario con un botón que muestra un mensaje

Este programa es una introducción básica que demuestra cómo mostrar un simple cuadro de mensaje (`MessageBox`) al hacer clic en un botón.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnMensaje`, **Text:** "Mostrar Mensaje".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnMensaje_Click(sender As Object, e As EventArgs) Handles
btnMensaje.Click
        MessageBox.Show("Hola, este es un mensaje")
    End Sub
End Class
```

- **Explicación:**
  - El código se encuentra dentro de la clase `Form1`.
  - El evento `btnMensaje_Click` se activa cuando el usuario hace clic en el botón `btnMensaje`.
  - Dentro de este evento, `MessageBox.Show("Hola, este es un mensaje")` despliega una ventana emergente que muestra el texto "Hola, este es un mensaje".

Programa 2. Formulario con una etiqueta y un botón que cambia el texto

Este ejercicio ilustra cómo un botón puede interactuar con otros controles para modificar sus propiedades, en este caso, el texto de una etiqueta.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblTexto`, **Text:** "Texto inicial".
  - `Button`: **Name:** `btnCambiar`, **Text:** "Cambiar Texto".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnCambiar_Click(sender As Object, e As EventArgs) Handles
btnCambiar.Click
        lblTexto.Text = "Texto cambiado"
    End Sub
End Class
```

- **Explicación:**
  - Cuando se hace clic en el botón `btnCambiar`, el evento `btnCambiar_Click` se ejecuta.
  - La línea `lblTexto.Text = "Texto cambiado"` actualiza la propiedad `Text` del control `lblTexto` (la etiqueta) a "Texto cambiado", haciendo que el texto visible en el formulario cambie.

Programa 3. Formulario con un cuadro de texto y un botón para mostrar su contenido

Este programa enseña cómo capturar la entrada del usuario desde un cuadro de texto y mostrarla en un `MessageBox`.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtEntrada`.
  - `Button`: **Name:** `btnMostrar`, **Text:** "Mostrar".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnMostrar_Click(sender As Object, e As EventArgs) Handles
btnMostrar.Click
        MessageBox.Show("Contenido: " & txtEntrada.Text)
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnMostrar`, el evento `btnMostrar_Click` se dispara.
  - `MessageBox.Show("Contenido: " & txtEntrada.Text)` toma el texto actual del `TextBox` `txtEntrada` (usando `txtEntrada.Text`) y lo concatena con la cadena "Contenido: ", mostrando el resultado en un cuadro de mensaje.

Programa 4. Formulario con dos botones que cambian el color de fondo

Este ejercicio demuestra cómo modificar las propiedades visuales del formulario, específicamente su color de fondo, mediante la interacción con botones.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnRojo`, **Text:** "Fondo Rojo".
  - `Button`: **Name:** `btnAzul`, **Text:** "Fondo Azul".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnRojo_Click(sender As Object, e As EventArgs) Handles
btnRojo.Click
        Me.BackColor = Color.Red
    End Sub
    Private Sub btnAzul_Click(sender As Object, e As EventArgs) Handles
btnAzul.Click
        Me.BackColor = Color.Blue
    End Sub
End Class
```

- **Explicación:**
  - El evento `btnRojo_Click` se activa cuando se hace clic en `btnRojo`, y `Me.BackColor = Color.Red` cambia el color de fondo del formulario (representado por `Me`) a rojo.
  - De manera similar, el evento `btnAzul_Click` cambia el color de fondo del formulario a azul cuando se hace clic en `btnAzul`.

Programa 5. Formulario con un cuadro de texto que convierte a mayúsculas

Este programa muestra la manipulación de cadenas de texto ingresadas por el usuario, convirtiéndolas a mayúsculas.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtEntrada`.
  - `Button`: **Name:** `btnMayusculas`, **Text:** "Convertir".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnMayusculas_Click(sender As Object, e As EventArgs)
Handles btnMayusculas.Click
        txtEntrada.Text = txtEntrada.Text.ToUpper()
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnMayusculas`, el evento `btnMayusculas_Click` se ejecuta.
  - `txtEntrada.Text.ToUpper()` convierte el texto actual del `TextBox` a mayúsculas, y luego `txtEntrada.Text = ...` asigna este nuevo texto (en mayúsculas) de vuelta al `TextBox`, actualizando lo que ve el usuario.

Programa 6. Formulario con dos cuadros de texto para sumar números

Este ejercicio introduce la captura de múltiples entradas numéricas, su conversión y la realización de una operación aritmética básica (suma), mostrando el resultado en una etiqueta.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtNum1`.
  - `TextBox`: **Name:** `txtNum2`.
  - `Button`: **Name:** `btnSumar`, **Text:** "Sumar".
  - `Label`: **Name:** `lblResultado`, **Text:** "Resultado:".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnSumar_Click(sender As Object, e As EventArgs) Handles
btnSumar.Click
        Dim a As Double = Val(txtNum1.Text)
        Dim b As Double = Val(txtNum2.Text)
        lblResultado.Text = "Resultado: " & (a + b).ToString()
    End Sub
End Class
```

- **Explicación:**
  - Cuando se hace clic en `btnSumar`, el evento `btnSumar_Click` se activa.
  - `Val(txtNum1.Text)` y `Val(txtNum2.Text)` convierten el texto de los `TextBox` a valores numéricos (`Double`).
  - Estos valores se suman (`a + b`), y el resultado se convierte a cadena (`.ToString()`) para ser mostrado junto con "Resultado: " en `lblResultado.Text`.

Programa 7. Formulario con un botón para limpiar cuadros de texto

Este programa enseña cómo limpiar el contenido de varios cuadros de texto a la vez.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtUno`.
  - `TextBox`: **Name:** `txtDos`.
  - `Button`: **Name:** `btnLimpiar`, **Text:** "Limpiar".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnLimpiar_Click(sender As Object, e As EventArgs) Handles
btnLimpiar.Click
        txtUno.Clear()
        txtDos.Clear()
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnLimpiar`, el evento `btnLimpiar_Click` se ejecuta.
  - Los métodos `Clear()` de `txtUno` y `txtDos` eliminan todo el texto de ambos `TextBox`s, dejándolos vacíos.

Programa 8. Formulario con un botón que cierra la aplicación

Un programa muy simple pero esencial que muestra cómo terminar la ejecución de una aplicación.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnSalir`, **Text:** "Salir".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles
btnSalir.Click
        Application.Exit()
    End Sub
End Class
```

- **Explicación:**
  - Cuando el usuario hace clic en `btnSalir`, el evento `btnSalir_Click` se activa.
  - `Application.Exit()` detiene todos los mensajes y finaliza la ejecución de la aplicación, cerrando el formulario.

Programa 9. Formulario con una etiqueta que muestra la hora actual

Este programa introduce el uso del control `Timer` para actualizar periódicamente un `Label` con la hora actual, mostrando un concepto de interactividad en tiempo real.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblHora`, **Text:** "Hora:".
  - `Timer`: **Name:** `Timer1`, **Enabled:** `True`, **Interval:** `1000`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
        lblHora.Text = "Hora: " & DateTime.Now.ToLongTimeString()
    End Sub
End Class
```

- **Explicación:**
  - El `Timer1` está habilitado (`Enabled: True`) y configurado para disparar su evento `Tick` cada 1000 milisegundos (1 segundo).
  - En cada `Timer1_Tick`, `DateTime.Now.ToLongTimeString()` obtiene la hora actual en un formato de cadena largo.
  - Esta cadena se asigna a `lblHora.Text`, actualizando la etiqueta cada segundo.

Programa 10. Formulario con un cuadro de texto que cuenta caracteres

Este ejercicio demuestra cómo reaccionar a los cambios en el texto de un `TextBox` y mostrar información en tiempo real, como el número de caracteres.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtEntrada`, **Multiline:** `True`.
  - `Label`: **Name:** `lblContar`, **Text:** "Caracteres: 0".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub txtEntrada_TextChanged(sender As Object, e As EventArgs)
Handles txtEntrada.TextChanged
        lblContar.Text = "Caracteres: " &
txtEntrada.Text.Length.ToString()
    End Sub
End Class
```

- **Explicación:**
  - El evento `txtEntrada_TextChanged` se dispara cada vez que el texto dentro de `txtEntrada` cambia.
  - `txtEntrada.Text.Length` obtiene la cantidad de caracteres en el `TextBox`.
  - Este valor se convierte a cadena (`.ToString()`) y se asigna a `lblContar.Text`, actualizando el contador en la etiqueta.

Programa 11. Formulario con dos botones para mostrar imágenes diferentes

Este programa enseña cómo cargar y mostrar imágenes desde archivos locales usando el control `PictureBox`.

- **Controles necesarios en el formulario:**
  - `PictureBox`: **Name:** `picImagen`, **SizeMode:** `StretchImage`, **Size:** `200x200`.
  - `Button`: **Name:** `btnImagen1`, **Text:** "Imagen 1".
  - `Button`: **Name:** `btnImagen2`, **Text:** "Imagen 2".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnImagen1_Click(sender As Object, e As EventArgs) Handles
btnImagen1.Click
        picImagen.Image = Image.FromFile("imagen1.jpg")
    End Sub
    Private Sub btnImagen2_Click(sender As Object, e As EventArgs) Handles
btnImagen2.Click
        picImagen.Image = Image.FromFile("imagen2.jpg")
    End Sub
End Class
```

- **Explicación:**
  - Cuando se hace clic en `btnImagen1`, `Image.FromFile("imagen1.jpg")` carga la imagen del archivo "imagen1.jpg" y la asigna a la propiedad `Image` de `picImagen`.
  - De forma similar, `btnImagen2_Click` carga y muestra "imagen2.jpg". (Se asume que "imagen1.jpg" y "imagen2.jpg" están en la misma carpeta que el ejecutable del programa o se especifica una ruta completa).

Programa 12. Formulario con un botón que cambia el tamaño de la ventana

Este ejercicio demuestra cómo modificar las dimensiones del formulario programáticamente.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnCambiarTamaño`, **Text:** "Cambiar Tamaño".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnCambiarTamaño_Click(sender As Object, e As EventArgs)
Handles btnCambiarTamaño.Click
        Me.Size = New Size(800, 600)
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnCambiarTamaño`, el evento `btnCambiarTamaño_Click` se ejecuta.
  - `Me.Size = New Size(800, 600)` establece el ancho del formulario a 800 píxeles y la altura a 600 píxeles.

Programa 13. Formulario con un cuadro de texto que solo permite números

Este programa enseña a validar la entrada del usuario en tiempo real, restringiendo el `TextBox` para que solo acepte caracteres numéricos.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtNumeros`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub txtNumeros_KeyPress(sender As Object, e As
KeyPressEventArgs) Handles txtNumeros.KeyPress
        If Not Char.IsControl(e.KeyChar) AndAlso Not
Char.IsDigit(e.KeyChar) Then
            e.Handled = True
        End If
    End Sub
End Class
```

- **Explicación:**
  - El evento `txtNumeros_KeyPress` se dispara cada vez que una tecla es presionada mientras el `TextBox` tiene el foco.
  - `Char.IsControl(e.KeyChar)` verifica si la tecla es una tecla de control (como Backspace, Enter).
  - `Char.IsDigit(e.KeyChar)` verifica si la tecla es un dígito numérico.
  - Si la tecla NO es un control Y NO es un dígito, `e.Handled = True` indica que el evento ha sido manejado y el carácter no se procesará, evitando que aparezca en el `TextBox`.

Programa 14. Formulario con tres botones que cambian el color de texto

Este ejercicio demuestra cómo cambiar el color de la fuente de un `Label` usando múltiples botones.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblTexto`, **Text:** "Texto de ejemplo".
  - `Button`: **Name:** `btnRojo`, **Text:** "Rojo".
  - `Button`: **Name:** `btnVerde`, **Text:** "Verde".
  - `Button`: **Name:** `btnAzul`, **Text:** "Azul".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnRojo_Click(sender As Object, e As EventArgs) Handles
btnRojo.Click
        lblTexto.ForeColor = Color.Red
    End Sub
    Private Sub btnVerde_Click(sender As Object, e As EventArgs) Handles
btnVerde.Click
        lblTexto.ForeColor = Color.Green
    End Sub
    Private Sub btnAzul_Click(sender As Object, e As EventArgs) Handles
btnAzul.Click
        lblTexto.ForeColor = Color.Blue
    End Sub
End Class
```

- **Explicación:**
  - Cada botón tiene un evento `_Click` asociado que modifica la propiedad `ForeColor` del `lblTexto` al color correspondiente (`Color.Red`, `Color.Green`, `Color.Blue`).

Programa 15. Formulario con un cuadro de texto y un botón para invertir texto

Este programa introduce la manipulación de cadenas para invertir su orden.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtEntrada`.
  - `Button`: **Name:** `btnInvertir`, **Text:** "Invertir".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnInvertir_Click(sender As Object, e As EventArgs)
Handles btnInvertir.Click
        Dim texto As String = txtEntrada.Text
        Dim invertido As String = StrReverse(texto)
        txtEntrada.Text = invertido
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnInvertir`, el evento `btnInvertir_Click` se ejecuta.
  - La función `StrReverse()` toma el texto de `txtEntrada` y devuelve una nueva cadena con los caracteres en orden inverso.
  - Este texto invertido se asigna de nuevo a `txtEntrada.Text`, actualizando el contenido del cuadro de texto.

Programa 16. Formulario con un contador de clics

Este ejercicio demuestra cómo mantener y actualizar un estado (un contador) dentro del formulario.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblContador`, **Text:** "Clics: 0".
  - `Button`: **Name:** `btnContar`, **Text:** "Contar Clic".
- **Código VB.NET:**

```
Public Class Form1
    Dim contador As Integer = 0
    Private Sub btnContar_Click(sender As Object, e As EventArgs) Handles
btnContar.Click
        contador += 1
        lblContador.Text = "Clics: " & contador.ToString()
    End Sub
End Class
```

- **Explicación:**
  - Se declara una variable `contador` de tipo `Integer` e inicializada a 0 a nivel de clase, para que su valor persista entre clics.
  - Cada vez que se hace clic en `btnContar`, el evento `btnContar_Click` incrementa `contador` en 1.
  - Luego, `lblContador.Text` se actualiza para mostrar el nuevo valor del contador.

Programa 17. Formulario con dos botones para mostrar mensajes distintos

Similar al Programa 1, este ejercicio refuerza el concepto de eventos de botón que desencadenan acciones diferentes.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnMensaje1`, **Text:** "Mensaje 1".
  - `Button`: **Name:** `btnMensaje2`, **Text:** "Mensaje 2".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnMensaje1_Click(sender As Object, e As EventArgs)
Handles btnMensaje1.Click
        MessageBox.Show("Este es el primer mensaje")
    End Sub
    Private Sub btnMensaje2_Click(sender As Object, e As EventArgs)
Handles btnMensaje2.Click
        MessageBox.Show("Este es el segundo mensaje")
    End Sub
End Class
```

- **Explicación:**
  - Cada botón, al ser clicado, muestra un `MessageBox` con un mensaje específico, diferenciando las acciones por evento.

Programa 18. Formulario con un CheckBox que habilita o deshabilita un botón

Este programa ilustra cómo un control (`CheckBox`) puede modificar el estado de otro control (`Button`), afectando su interacción.

- **Controles necesarios en el formulario:**
  - `CheckBox`: **Name:** `chkHabilitar`, **Text:** "Habilitar botón".
  - `Button`: **Name:** `btnAccion`, **Text:** "Acción", **Enabled:** `False`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub chkHabilitar_CheckedChanged(sender As Object, e As
EventArgs) Handles chkHabilitar.CheckedChanged
        btnAccion.Enabled = chkHabilitar.Checked
    End Sub
End Class
```

- **Explicación:**
  - El evento `chkHabilitar_CheckedChanged` se dispara cada vez que el estado de marcado del `CheckBox` cambia.
  - `chkHabilitar.Checked` devuelve `True` si el `CheckBox` está marcado y `False` si no lo está.
  - Esta propiedad se asigna directamente a `btnAccion.Enabled`, de modo que el botón se habilita o deshabilita según el estado del `CheckBox`.

Programa 19. Formulario con un TextBox multilinea y un botón para contar líneas

Este ejercicio muestra la manipulación de texto en un `TextBox` multilinea y cómo obtener información estructurada de él, como el número de líneas.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtMultilinea`, **Multiline:** `True`, **Size:** `200x100`.
  - `Button`: **Name:** `btnContar`, **Text:** "Contar Líneas".
  - `Label`: **Name:** `lblLineas`, **Text:** "Líneas: 0".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnContar_Click(sender As Object, e As EventArgs) Handles
btnContar.Click
        Dim lineas As Integer = txtMultilinea.Lines.Length
        lblLineas.Text = "Líneas: " & lineas.ToString()
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnContar`, el evento `btnContar_Click` se activa.
  - `txtMultilinea.Lines` devuelve un array de cadenas, donde cada elemento es una línea del texto del `TextBox`.
  - `.Length` de este array proporciona el número total de líneas, que luego se muestra en `lblLineas.Text`.

Programa 20. Formulario con una etiqueta que cambia de color al pasar el mouse

Este programa introduce los eventos relacionados con el mouse, permitiendo una interactividad visual dinámica con un control.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblColor`, **Text:** "Pasa el mouse por aquí".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub lblColor_MouseEnter(sender As Object, e As EventArgs)
Handles lblColor.MouseEnter
        lblColor.ForeColor = Color.Red
    End Sub
    Private Sub lblColor_MouseLeave(sender As Object, e As EventArgs)
Handles lblColor.MouseLeave
        lblColor.ForeColor = Color.Black
    End Sub
End Class
```

- **Explicación:**
  - El evento `lblColor_MouseEnter` se dispara cuando el puntero del mouse entra en el área del `lblColor`, cambiando su color de fuente a rojo.
  - El evento `lblColor_MouseLeave` se dispara cuando el puntero del mouse sale del área del `lblColor`, restaurando su color de fuente a negro.

Programa 21. Formulario con un botón que cambia el ícono del formulario

Este ejercicio demuestra cómo modificar el ícono de la ventana del formulario en tiempo de ejecución.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnIcono`, **Text:** "Cambiar Ícono".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnIcono_Click(sender As Object, e As EventArgs) Handles
btnIcono.Click
        Me.Icon = New Icon("icono.ico")
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnIcono`, el evento `btnIcono_Click` se ejecuta.
  - `Me.Icon = New Icon("icono.ico")` crea un nuevo objeto `Icon` a partir del archivo "icono.ico" (que debe existir en la misma ruta que el ejecutable o una ruta especificada) y lo asigna a la propiedad `Icon` del formulario, cambiando el ícono visible en la barra de título y la barra de tareas.

Programa 22. Formulario con un TrackBar que controla el tamaño del texto

Este programa introduce el control `TrackBar` para permitir al usuario ajustar un valor de forma visual, en este caso, el tamaño de la fuente de un texto.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblTexto`, **Text:** "Texto de ejemplo".
  - `TrackBar`: **Name:** `trkTamaño`, **Minimum:** `8`, **Maximum:** `40`, **Value:** `12`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub trkTamaño_Scroll(sender As Object, e As EventArgs) Handles
trkTamaño.Scroll
        lblTexto.Font = New Font(lblTexto.Font.FontFamily,
trkTamaño.Value)
    End Sub
End Class
```

- **Explicación:**
  - El evento `trkTamaño_Scroll` se dispara cuando el usuario mueve el deslizador del `TrackBar`.
  - `trkTamaño.Value` obtiene el valor actual del `TrackBar`.
  - Se crea un nuevo objeto `Font` para `lblTexto`, manteniendo la misma familia de fuente pero utilizando el valor del `TrackBar` para el tamaño de la fuente, lo que hace que el texto del `Label` aumente o disminuya de tamaño.

Programa 23. Formulario con un cuadro de texto que valida correo electrónico

Este ejercicio introduce el concepto de validación de datos usando expresiones regulares para verificar si una cadena tiene el formato de un correo electrónico.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtCorreo`.
  - `Button`: **Name:** `btnValidar`, **Text:** "Validar".
  - `Label`: **Name:** `lblResultado`, **Text:** "".
- **Código VB.NET:**

```
Imports System.Text.RegularExpressions
Public Class Form1
    Private Sub btnValidar_Click(sender As Object, e As EventArgs) Handles
btnValidar.Click
        Dim patron As String = "^[^@\s]+@[^@\s]+\.[^@\s]+$"
        If Regex.IsMatch(txtCorreo.Text, patron) Then
            lblResultado.Text = "Correo válido"
        Else
            lblResultado.Text = "Correo inválido"
        End If
    End Sub
End Class
```

- **Explicación:**
  - Se importa el espacio de nombres `System.Text.RegularExpressions`.
  - Cuando se hace clic en `btnValidar`, se define un `patron` de expresión regular para un formato de correo electrónico.
  - `Regex.IsMatch(txtCorreo.Text, patron)` verifica si el texto ingresado en `txtCorreo` coincide con el patrón.
  - `lblResultado.Text` muestra si el correo es "válido" o "inválido" en función de la coincidencia.

Programa 24. Formulario con un botón que rota imágenes

Este programa demuestra cómo manipular imágenes, específicamente rotarlas, utilizando métodos de la clase `Image`.

- **Controles necesarios en el formulario:**
  - `PictureBox`: **Name:** `picImagen`, **SizeMode:** `StretchImage`, **Size:** `200x200`.
  - `Button`: **Name:** `btnRotar`, **Text:** "Rotar".
- **Código VB.NET:**

```
Public Class Form1
    Dim imagen As Image = Image.FromFile("imagen.jpg")
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        picImagen.Image = imagen
    End Sub
    Private Sub btnRotar_Click(sender As Object, e As EventArgs) Handles
btnRotar.Click
        imagen.RotateFlip(RotateFlipType.Rotate90FlipNone)
        picImagen.Image = imagen
    End Sub
End Class
```

- **Explicación:**
  - Se carga una imagen desde "imagen.jpg" y se almacena en la variable a nivel de clase `imagen`.
  - En `Form1_Load`, la imagen se muestra inicialmente en `picImagen`.
  - Cuando se hace clic en `btnRotar`, el método `RotateFlip(RotateFlipType.Rotate90FlipNone)` rota la imagen almacenada 90 grados en el sentido de las agujas del reloj sin voltearla.
  - Luego, `picImagen.Image = imagen` actualiza el `PictureBox` con la imagen rotada.

Programa 25. Formulario con una lista de opciones en un ComboBox

Este ejercicio introduce el control `ComboBox` para presentar una lista desplegable de opciones al usuario.

- **Controles necesarios en el formulario:**
  - `ComboBox`: **Name:** `cmbOpciones`, **DropDownStyle:** `DropDownList`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        cmbOpciones.Items.Add("Opción 1")
        cmbOpciones.Items.Add("Opción 2")
        cmbOpciones.Items.Add("Opción 3")
    End Sub
End Class
```

- **Explicación:**
  - En el evento `Form1_Load`, que se dispara cuando el formulario se carga, se añaden elementos (cadenas de texto) a la colección `Items` de `cmbOpciones`. Esto llena la lista desplegable con las opciones que el usuario puede seleccionar.

Programa 26. Formulario con un ListBox que agrega elementos desde un TextBox

Este programa muestra cómo el usuario puede añadir elementos a una lista dinámica (`ListBox`) a partir del texto ingresado en un `TextBox`.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtElemento`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar".
  - `ListBox`: **Name:** `lstElementos`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
        If txtElemento.Text <> "" Then
            lstElementos.Items.Add(txtElemento.Text)
            txtElemento.Clear()
        End If
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnAgregar`, el evento `btnAgregar_Click` verifica si `txtElemento` no está vacío.
  - Si hay texto, se añade como un nuevo elemento a `lstElementos.Items`.
  - Luego, `txtElemento.Clear()` vacía el `TextBox` para una nueva entrada.

Programa 27. Formulario con un botón que elimina el elemento seleccionado del ListBox

Este ejercicio complementa el anterior, mostrando cómo interactuar con los elementos de un `ListBox` para eliminarlos.

- **Controles necesarios en el formulario:**
  - `ListBox`: **Name:** `lstElementos`.
  - `Button`: **Name:** `btnEliminar`, **Text:** "Eliminar Seleccionado".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnEliminar_Click(sender As Object, e As EventArgs)
Handles btnEliminar.Click
        If lstElementos.SelectedIndex >= 0 Then
            lstElementos.Items.RemoveAt(lstElementos.SelectedIndex)
        End If
    End Sub
End Class
```

- **Explicación:**
  - Cuando se hace clic en `btnEliminar`, el evento `btnEliminar_Click` verifica si hay algún elemento seleccionado en `lstElementos` (si `SelectedIndex` es mayor o igual a 0).
  - Si hay un elemento seleccionado, `lstElementos.Items.RemoveAt(lstElementos.SelectedIndex)` elimina el elemento en la posición seleccionada.

Programa 28. Formulario con un cuadro de texto que convierte a número romano

Este programa es un ejemplo de conversión de formatos numéricos, aplicando lógica para transformar un número entero a su representación romana.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtNumero`.
  - `Button`: **Name:** `btnConvertir`, **Text:** "Convertir".
  - `Label`: **Name:** `lblRomano`, **Text:** "".
- **Código VB.NET:**

```
Public Class Form1
    Private Function ConvertirRomano(num As Integer) As String
        Dim valores() As Integer = {1000, 900, 500, 400, 100, 90, 50, 40,
10, 9, 5, 4, 1}
        Dim simbolos() As String = {"M", "CM", "D", "CD", "C", "XC", "L",
"XL", "X", "IX", "V", "IV", "I"}
        Dim resultado As String = ""
        For i As Integer = 0 To valores.Length - 1
            While num >= valores(i)
                num -= valores(i)
                resultado &= simbolos(i)
            End While
        Next
        Return resultado
    End Function
    Private Sub btnConvertir_Click(sender As Object, e As EventArgs)
Handles btnConvertir.Click
        Dim numero As Integer
        If Integer.TryParse(txtNumero.Text, numero) Then
            lblRomano.Text = ConvertirRomano(numero)
        Else
            lblRomano.Text = "Número inválido"
        End If
    End Sub
End Class
```

- **Explicación:**
  - La función `ConvertirRomano` toma un entero y, utilizando arrays de `valores` y `simbolos`, construye la representación romana mediante restas sucesivas del número.
  - Cuando se hace clic en `btnConvertir`, `Integer.TryParse` intenta convertir el texto de `txtNumero` a un entero.
  - Si la conversión es exitosa, llama a `ConvertirRomano` y muestra el resultado en `lblRomano.Text`. De lo contrario, indica que el número es inválido.

Programa 29. Formulario con un botón que genera un número aleatorio

Este programa introduce el uso de números aleatorios para generar resultados impredecibles.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnAleatorio`, **Text:** "Generar".
  - `Label`: **Name:** `lblNumero`, **Text:** "".
- **Código VB.NET:**

```
Public Class Form1
    Dim rnd As New Random()
    Private Sub btnAleatorio_Click(sender As Object, e As EventArgs)
Handles btnAleatorio.Click
        lblNumero.Text = "Número: " & rnd.Next(1, 101).ToString()
    End Sub
End Class
```

- **Explicación:**
  - Se crea una instancia de la clase `Random` a nivel de clase para generar números aleatorios.
  - Al hacer clic en `btnAleatorio`, `rnd.Next(1, 101)` genera un número entero aleatorio entre 1 (inclusive) y 101 (exclusivo), es decir, de 1 a 100.
  - Este número se convierte a cadena y se muestra en `lblNumero.Text`.

Programa 30. Formulario con un temporizador que actualiza la hora

Este programa es idéntico al Programa 9, reforzando la implementación de un `Timer` para actualizar información en intervalos regulares.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblHora`, **Text:** "Hora: ".
  - `Timer`: **Name:** `Timer1`, **Enabled:** `True`, **Interval:** `1000`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
        lblHora.Text = "Hora: " & DateTime.Now.ToLongTimeString()
    End Sub
End Class
```

- **Explicación:**
  - El `Timer1` está habilitado y se dispara cada 1000 milisegundos.
  - En cada `Timer1_Tick`, `DateTime.Now.ToLongTimeString()` obtiene la hora actual y la asigna a `lblHora.Text`, actualizando el texto de la etiqueta cada segundo.

Programa 31. Formulario con un botón que oculta y muestra una etiqueta

Este programa demuestra cómo controlar la visibilidad de un control en el formulario con un simple botón.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblTexto`, **Text:** "Soy una etiqueta".
  - `Button`: **Name:** `btnOcultarMostrar`, **Text:** "Ocultar/Mostrar".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnOcultarMostrar_Click(sender As Object, e As EventArgs)
Handles btnOcultarMostrar.Click
        lblTexto.Visible = Not lblTexto.Visible
    End Sub
End Class
```

- **Explicación:**
  - El evento `btnOcultarMostrar_Click` se activa al hacer clic en el botón.
  - La línea `lblTexto.Visible = Not lblTexto.Visible` alterna el valor de la propiedad `Visible` de la etiqueta `lblTexto`. Si `Visible` es `True`, se convierte en `False` (ocultando la etiqueta), y si es `False`, se convierte en `True` (mostrándola).

Programa 32. Formulario con un botón que abre un nuevo formulario vacío

Este programa introduce el concepto de múltiples formularios en una aplicación, permitiendo abrir una nueva ventana con un botón.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnAbrir`, **Text:** "Abrir Formulario".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnAbrir_Click(sender As Object, e As EventArgs) Handles
btnAbrir.Click
        Dim nuevoForm As New Form
        nuevoForm.Text = "Nuevo Formulario"
        nuevoForm.Show()
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnAbrir`, se ejecuta el evento `btnAbrir_Click`.
  - `Dim nuevoForm As New Form` crea una nueva instancia de un formulario genérico.
  - `nuevoForm.Text = "Nuevo Formulario"` establece el título de la nueva ventana.
  - `nuevoForm.Show()` hace visible este nuevo formulario en la pantalla.

Programa 33. Formulario con una barra de progreso que avanza al presionar un botón

Este ejercicio utiliza el control `ProgressBar` para visualizar el progreso de una tarea simulada.

- **Controles necesarios en el formulario:**
  - `ProgressBar`: **Name:** `prgBarra`, **Minimum:** `0`, **Maximum:** `100`, **Value:** `0`.
  - `Button`: **Name:** `btnAvanzar`, **Text:** "Avanzar".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnAvanzar_Click(sender As Object, e As EventArgs) Handles
btnAvanzar.Click
        If prgBarra.Value < prgBarra.Maximum Then
            prgBarra.Value += 10
        End If
    End Sub
End Class
```

- **Explicación:**
  - El evento `btnAvanzar_Click` se dispara al hacer clic en el botón.
  - La condición `If prgBarra.Value < prgBarra.Maximum` asegura que la barra de progreso no exceda su valor máximo (100).
  - `prgBarra.Value += 10` incrementa el valor de la barra de progreso en 10 unidades cada vez que se presiona el botón, visualizando un avance.

Programa 34. Formulario con un botón que copia texto al portapapeles

Este programa interactúa con el sistema operativo para copiar texto a la memoria del portapapeles.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtEntrada`.
  - `Button`: **Name:** `btnCopiar`, **Text:** "Copiar".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnCopiar_Click(sender As Object, e As EventArgs) Handles
btnCopiar.Click
        Clipboard.SetText(txtEntrada.Text)
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnCopiar`, el evento `btnCopiar_Click` se ejecuta.
  - `Clipboard.SetText(txtEntrada.Text)` toma el texto del `TextBox` `txtEntrada` y lo coloca en el portapapeles del sistema, haciéndolo disponible para pegarlo en otras aplicaciones.

Programa 35. Formulario con un cuadro de texto que limita caracteres

Este programa demuestra una forma sencilla de restringir la cantidad de texto que un usuario puede ingresar en un `TextBox`.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtLimitado`, **MaxLength:** `10`.
  - `Label`: **Name:** `lblAviso`, **Text:** "Máximo 10 caracteres".
- **Código VB.NET:**

```
Public Class Form1
    ' (Nota: El límite se aplica directamente con la propiedad MaxLength
del TextBox, no se necesita
    ' código extra.)
End Class
```

- **Explicación:**
  - La clave de este programa no reside en el código, sino en la propiedad `MaxLength` del control `TextBox`.
  - Al establecer `MaxLength` a `10` en el diseñador de Visual Studio, el `TextBox` automáticamente impedirá que el usuario ingrese más de 10 caracteres, sin necesidad de escribir código adicional. El `Label` sirve como una indicación visual para el usuario.

Programa 36. Formulario con un botón que abre un cuadro de diálogo de color

Este ejercicio utiliza el control `ColorDialog` para permitir al usuario seleccionar un color y aplicarlo a un control del formulario.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnColor`, **Text:** "Elegir Color".
  - `Label`: **Name:** `lblTexto`, **Text:** "Texto de ejemplo".
  - `ColorDialog`: **Name:** `ColorDialog1`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnColor_Click(sender As Object, e As EventArgs) Handles
btnColor.Click
        If ColorDialog1.ShowDialog() = DialogResult.OK Then
            lblTexto.ForeColor = ColorDialog1.Color
        End If
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnColor`, se dispara el evento `btnColor_Click`.
  - `ColorDialog1.ShowDialog()` abre la ventana estándar de selección de color de Windows. Si el usuario selecciona un color y hace clic en "Aceptar" (`DialogResult.OK`), la condición se cumple.
  - `lblTexto.ForeColor = ColorDialog1.Color` asigna el color elegido por el usuario (disponible en `ColorDialog1.Color`) a la propiedad `ForeColor` de `lblTexto`, cambiando el color del texto.

Programa 37. Formulario con un botón que abre un cuadro de diálogo de fuente

Similar al `ColorDialog`, este programa utiliza `FontDialog` para permitir al usuario elegir una fuente y aplicarla a una etiqueta.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnFuente`, **Text:** "Elegir Fuente".
  - `Label`: **Name:** `lblTexto`, **Text:** "Texto de ejemplo".
  - `FontDialog`: **Name:** `FontDialog1`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnFuente_Click(sender As Object, e As EventArgs) Handles
btnFuente.Click
        If FontDialog1.ShowDialog() = DialogResult.OK Then
            lblTexto.Font = FontDialog1.Font
        End If
    End Sub
End Class
```

- **Explicación:**
  - El evento `btnFuente_Click` se activa al presionar el botón.
  - `FontDialog1.ShowDialog()` abre el cuadro de diálogo estándar de fuentes. Si el usuario selecciona una fuente y hace clic en "Aceptar", la propiedad `Font` de `lblTexto` se actualiza con la fuente elegida (`FontDialog1.Font`).

Programa 38. Formulario con un botón que abre un cuadro de diálogo de archivo

Este programa permite al usuario seleccionar un archivo del sistema mediante el `OpenFileDialog`.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnArchivo`, **Text:** "Abrir Archivo".
  - `OpenFileDialog`: **Name:** `OpenFileDialog1`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnArchivo_Click(sender As Object, e As EventArgs) Handles
btnArchivo.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            MessageBox.Show("Archivo seleccionado: " &
OpenFileDialog1.FileName)
        End If
    End Sub
End Class
```

- **Explicación:**
  - Cuando se hace clic en `btnArchivo`, el evento `btnArchivo_Click` se ejecuta.
  - `OpenFileDialog1.ShowDialog()` abre el cuadro de diálogo para seleccionar un archivo. Si el usuario selecciona un archivo y pulsa "Abrir", la condición se cumple.
  - `OpenFileDialog1.FileName` contiene la ruta completa del archivo seleccionado, que se muestra en un `MessageBox`.

Programa 39. Formulario con un botón que abre un cuadro de diálogo de carpeta

Este programa utiliza el control `FolderBrowserDialog` para permitir al usuario seleccionar una carpeta del sistema.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnCarpeta`, **Text:** "Elegir Carpeta".
  - `FolderBrowserDialog`: **Name:** `FolderBrowserDialog1`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnCarpeta_Click(sender As Object, e As EventArgs) Handles
btnCarpeta.Click
        If FolderBrowserDialog1.ShowDialog() = DialogResult.OK Then
            MessageBox.Show("Carpeta seleccionada: " &
FolderBrowserDialog1.SelectedPath)
        End If
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnCarpeta`, el evento `btnCarpeta_Click` se activa.
  - `FolderBrowserDialog1.ShowDialog()` abre el cuadro de diálogo para seleccionar una carpeta. Si el usuario selecciona una carpeta y pulsa "Aceptar", la condición se cumple.
  - `FolderBrowserDialog1.SelectedPath` contiene la ruta de la carpeta elegida, que se muestra en un `MessageBox`.

Programa 40. Formulario con un menú simple con dos opciones

Este programa introduce el uso del control `MenuStrip` para crear un menú en la parte superior del formulario, con opciones que realizan acciones simples.

- **Controles necesarios en el formulario:**
  - `MenuStrip`: **Name:** `MenuStrip1`.
  - `ToolStripMenuItem`: **Name:** `opcion1ToolStripMenuItem`, **Text:** "Opción 1" (anidado bajo un menú principal, por ejemplo "Archivo").
  - `ToolStripMenuItem`: **Name:** `opcion2ToolStripMenuItem`, **Text:** "Opción 2" (anidado bajo el mismo menú principal).
- **Código VB.NET:**

```
Public Class Form1
    Private Sub opcion1ToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles opcion1ToolStripMenuItem.Click
        MessageBox.Show("Elegiste la Opción 1")
    End Sub
    Private Sub opcion2ToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles opcion2ToolStripMenuItem.Click
        MessageBox.Show("Elegiste la Opción 2")
    End Sub
End Class
```

- **Explicación:**
  - Cada `ToolStripMenuItem` tiene un evento `_Click` asociado.
  - Cuando el usuario hace clic en "Opción 1" o "Opción 2" del menú, el `MessageBox.Show` correspondiente se ejecuta, mostrando un mensaje específico para cada opción.

Programa 41. Formulario con un menú que cambia el color de fondo

Este programa expande el uso de `MenuStrip` para permitir al usuario cambiar el color de fondo del formulario a través de opciones de menú.

- **Controles necesarios en el formulario:**
  - `MenuStrip`: **Name:** `MenuStrip1`.
  - `ToolStripMenuItem`: **Name:** `rojoToolStripMenuItem`, **Text:** "Rojo".
  - `ToolStripMenuItem`: **Name:** `verdeToolStripMenuItem`, **Text:** "Verde".
  - `ToolStripMenuItem`: **Name:** `azulToolStripMenuItem`, **Text:** "Azul".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub rojoToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles rojoToolStripMenuItem.Click
        Me.BackColor = Color.Red
    End Sub
    Private Sub verdeToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles verdeToolStripMenuItem.Click
        Me.BackColor = Color.Green
    End Sub
    Private Sub azulToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles azulToolStripMenuItem.Click
        Me.BackColor = Color.Blue
    End Sub
End Class
```

- **Explicación:**
  - Cada `ToolStripMenuItem` de color tiene su propio evento `_Click`.
  - Al seleccionar una opción del menú, la propiedad `BackColor` del formulario (`Me.BackColor`) se establece en el color correspondiente (`Color.Red`, `Color.Green`, `Color.Blue`), cambiando el color de fondo del formulario.

Programa 42. Formulario con un menú que muestra un mensaje

Este es un ejemplo simple de cómo una opción de menú puede desencadenar una acción básica, como mostrar un mensaje.

- **Controles necesarios en el formulario:**
  - `MenuStrip`: **Name:** `MenuStrip1`.
  - `ToolStripMenuItem`: **Name:** `mensajeToolStripMenuItem`, **Text:** "Mostrar Mensaje".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub mensajeToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles mensajeToolStripMenuItem.Click
        MessageBox.Show("Este es un mensaje desde el menú")
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en el `mensajeToolStripMenuItem`, el evento `mensajeToolStripMenuItem_Click` se ejecuta, mostrando un `MessageBox` con el texto especificado.

Programa 43. Formulario con un menú para abrir otro formulario

Este programa integra la funcionalidad de abrir un nuevo formulario a través de una opción de menú, similar al Programa 32.

- **Controles necesarios en el formulario:**
  - `MenuStrip`: **Name:** `MenuStrip1`.
  - `ToolStripMenuItem`: **Name:** `abrirFormToolStripMenuItem`, **Text:** "Abrir Formulario".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub abrirFormToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles abrirFormToolStripMenuItem.Click
        Dim nuevoForm As New Form
        nuevoForm.Text = "Nuevo Formulario"
        nuevoForm.Show()
    End Sub
End Class
```

- **Explicación:**
  - Cuando se selecciona `abrirFormToolStripMenuItem`, se crea una nueva instancia de un formulario genérico, se le asigna un título y se muestra en pantalla, abriendo una nueva ventana de aplicación.

Programa 44. Formulario con una barra de herramientas (ToolStrip)

Este programa introduce el control `ToolStrip` para crear una barra de herramientas con botones que realizan acciones, similar a un menú pero con iconos y más accesibilidad visual.

- **Controles necesarios en el formulario:**
  - `ToolStrip`: **Name:** `ToolStrip1`.
  - `ToolStripButton`: **Name:** `btnNuevo`, **Text:** "Nuevo".
  - `ToolStripButton`: **Name:** `btnSalir`, **Text:** "Salir".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnNuevo_Click(sender As Object, e As EventArgs) Handles
btnNuevo.Click
        MessageBox.Show("Has pulsado 'Nuevo'")
    End Sub
    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles
btnSalir.Click
        Application.Exit()
    End Sub
End Class
```

- **Explicación:**
  - Cada `ToolStripButton` tiene su propio evento `_Click`.
  - El botón `btnNuevo` muestra un mensaje.
  - El botón `btnSalir` utiliza `Application.Exit()` para cerrar la aplicación, de forma idéntica al Programa 8.

Programa 45. Formulario con un cuadro de texto que busca en un ListBox

Este programa permite al usuario buscar un elemento específico dentro de un `ListBox` utilizando un `TextBox`.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtBuscar`.
  - `Button`: **Name:** `btnBuscar`, **Text:** "Buscar".
  - `ListBox`: **Name:** `lstElementos`. (Se recomienda poblarlo con algunos elementos previamente).
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnBuscar_Click(sender As Object, e As EventArgs) Handles
btnBuscar.Click
        Dim item As String = txtBuscar.Text
        Dim index As Integer = lstElementos.FindString(item)
        If index <> -1 Then
            lstElementos.SelectedIndex = index
        Else
            MessageBox.Show("Elemento no encontrado")
        End If
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnBuscar`, el evento `btnBuscar_Click` se ejecuta.
  - `lstElementos.FindString(item)` busca el texto ingresado en `txtBuscar.Text` dentro de `lstElementos` y devuelve el índice de la primera coincidencia. Si no encuentra nada, devuelve -1.
  - Si se encuentra el elemento (`index <> -1`), `lstElementos.SelectedIndex = index` lo selecciona en la lista. Si no, se muestra un `MessageBox` indicando que no se encontró.

Programa 46. Formulario con un botón que cambia el título del formulario

Este programa es un ejemplo básico de cómo modificar la propiedad `Text` del formulario mismo, que controla el texto de su barra de título.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnTitulo`, **Text:** "Cambiar Título".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnTitulo_Click(sender As Object, e As EventArgs) Handles
btnTitulo.Click
        Me.Text = "Nuevo Título del Formulario"
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnTitulo`, el evento `btnTitulo_Click` se dispara.
  - `Me.Text = "Nuevo Título del Formulario"` cambia la propiedad `Text` del formulario actual (`Me`) al nuevo texto, actualizando el título de la ventana.

Programa 47. Formulario con dos botones que mueven un Label en el formulario

Este ejercicio muestra la manipulación de la posición de un control en tiempo de ejecución, utilizando las propiedades de ubicación.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblMover`, **Text:** "Mueve este texto".
  - `Button`: **Name:** `btnIzquierda`, **Text:** "Izquierda".
  - `Button`: **Name:** `btnDerecha`, **Text:** "Derecha".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnIzquierda_Click(sender As Object, e As EventArgs)
Handles btnIzquierda.Click
        lblMover.Left -= 10
    End Sub
    Private Sub btnDerecha_Click(sender As Object, e As EventArgs) Handles
btnDerecha.Click
        lblMover.Left += 10
    End Sub
End Class
```

- **Explicación:**
  - `lblMover.Left` representa la coordenada X (posición horizontal) del lado izquierdo del `Label`.
  - Al hacer clic en `btnIzquierda`, `lblMover.Left -= 10` decrementa la posición horizontal en 10 píxeles, moviendo la etiqueta hacia la izquierda.
  - Al hacer clic en `btnDerecha`, `lblMover.Left += 10` incrementa la posición horizontal en 10 píxeles, moviendo la etiqueta hacia la derecha.

Programa 48. Formulario con un botón que muestra la fecha actual

Este programa utiliza la clase `DateTime` para obtener la fecha actual del sistema y mostrarla en un `Label`.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnFecha`, **Text:** "Mostrar Fecha".
  - `Label`: **Name:** `lblFecha`, **Text:** "".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnFecha_Click(sender As Object, e As EventArgs) Handles
btnFecha.Click
        lblFecha.Text = "Hoy es: " & DateTime.Now.ToLongDateString()
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnFecha`, el evento `btnFecha_Click` se ejecuta.
  - `DateTime.Now` obtiene la fecha y hora actuales del sistema. `ToLongDateString()` convierte esta fecha en una cadena de texto larga (por ejemplo, "lunes, 1 de enero de 2024").
  - El resultado se concatena con "Hoy es: " y se muestra en `lblFecha.Text`.

Programa 49. Formulario con un PictureBox que muestra diferentes imágenes

Este programa es idéntico al Programa 11, reforzando la carga y visualización de imágenes desde archivos locales en un `PictureBox`.

- **Controles necesarios en el formulario:**
  - `PictureBox`: **Name:** `picImagen`, **SizeMode:** `StretchImage`, **Size:** `200x200`.
  - `Button`: **Name:** `btnImagen1`, **Text:** "Imagen 1".
  - `Button`: **Name:** `btnImagen2`, **Text:** "Imagen 2".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnImagen1_Click(sender As Object, e As EventArgs) Handles
btnImagen1.Click
        picImagen.Image = Image.FromFile("imagen1.jpg")
    End Sub
    Private Sub btnImagen2_Click(sender As Object, e As EventArgs) Handles
btnImagen2.Click
        picImagen.Image = Image.FromFile("imagen2.jpg")
    End Sub
End Class
```

- **Explicación:**
  - Cada botón tiene un evento `_Click` que carga un archivo de imagen específico (`imagen1.jpg` o `imagen2.jpg`) utilizando `Image.FromFile()` y lo asigna a la propiedad `Image` del `PictureBox` `picImagen`. Se asume que los archivos de imagen están en la misma carpeta que el ejecutable de la aplicación.

Programa 50. Formulario con un botón que guarda texto en un archivo

Este programa introduce la funcionalidad de guardar datos (texto) en un archivo en el disco duro, utilizando el `SaveFileDialog`.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtContenido`, **Multiline:** `True`, **Size:** `200x100`.
  - `Button`: **Name:** `btnGuardar`, **Text:** "Guardar".
  - `SaveFileDialog`: **Name:** `SaveFileDialog1`.
- **Código VB.NET:**

```
Imports System.IO
Public Class Form1
    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click
        If SaveFileDialog1.ShowDialog() = DialogResult.OK Then
            File.WriteAllText(SaveFileDialog1.FileName, txtContenido.Text)
            MessageBox.Show("Archivo guardado con éxito")
        End If
    End Sub
End Class
```

- **Explicación:**
  - Se importa el espacio de nombres `System.IO` para las operaciones de archivo.
  - Al hacer clic en `btnGuardar`, el evento `btnGuardar_Click` se ejecuta.
  - `SaveFileDialog1.ShowDialog()` abre el cuadro de diálogo para que el usuario elija la ubicación y el nombre del archivo a guardar. Si el usuario hace clic en "Guardar" (`DialogResult.OK`), la condición se cumple.
  - `File.WriteAllText(SaveFileDialog1.FileName, txtContenido.Text)` escribe todo el texto de `txtContenido` en el archivo especificado por la ruta seleccionada. Finalmente, un `MessageBox` confirma el guardado.

Programa 51. Agenda de contactos básica con ListBox

Este programa es el primer ejercicio de la **Categoría 2 (Programas intermedios)** y crea una agenda simple que permite agregar nombres a un `ListBox`.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtNombre`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar".
  - `ListBox`: **Name:** `lstContactos`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
        If txtNombre.Text <> "" Then
            lstContactos.Items.Add(txtNombre.Text)
            txtNombre.Clear()
        End If
    End Sub
End Class
```

- **Explicación:**
  - Cuando se hace clic en `btnAgregar`, el evento `btnAgregar_Click` se ejecuta.
  - `If txtNombre.Text <> ""` verifica que el `TextBox` no esté vacío.
  - `lstContactos.Items.Add(txtNombre.Text)` añade el texto del `TextBox` como un nuevo elemento en el `ListBox`.
  - `txtNombre.Clear()` vacía el `TextBox` después de agregar el contacto, preparándolo para la siguiente entrada.

Programa 52. Calculadora simple (suma, resta, multiplicación, división)

Este programa implementa una calculadora básica con las cuatro operaciones aritméticas fundamentales.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtNum1`.
  - `TextBox`: **Name:** `txtNum2`.
  - `Button`: **Name:** `btnSumar`, **Text:** "Sumar".
  - `Button`: **Name:** `btnRestar`, **Text:** "Restar".
  - `Button`: **Name:** `btnMultiplicar`, **Text:** "Multiplicar".
  - `Button`: **Name:** `btnDividir`, **Text:** "Dividir".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnSumar_Click(sender As Object, e As EventArgs) Handles
btnSumar.Click
        lblResultado.Text = (Val(txtNum1.Text) +
Val(txtNum2.Text)).ToString()
    End Sub
    Private Sub btnRestar_Click(sender As Object, e As EventArgs) Handles
btnRestar.Click
        lblResultado.Text = (Val(txtNum1.Text) -
Val(txtNum2.Text)).ToString()
    End Sub
    Private Sub btnMultiplicar_Click(sender As Object, e As EventArgs)
Handles btnMultiplicar.Click
        lblResultado.Text = (Val(txtNum1.Text) *
Val(txtNum2.Text)).ToString()
    End Sub
    Private Sub btnDividir_Click(sender As Object, e As EventArgs) Handles
btnDividir.Click
        If Val(txtNum2.Text) <> 0 Then
            lblResultado.Text = (Val(txtNum1.Text) /
Val(txtNum2.Text)).ToString()
        Else
            lblResultado.Text = "Error: división por 0"
        End If
    End Sub
End Class
```

- **Explicación:**
  - Cada botón de operación tiene un evento `_Click` que:
    - Obtiene los valores numéricos de `txtNum1` y `txtNum2` usando `Val()`.
    - Realiza la operación correspondiente (suma, resta, multiplicación, división).
    - Convierte el resultado a cadena (`.ToString()`) y lo muestra en `lblResultado.Text`.
  - Para la división, se incluye una validación `If Val(txtNum2.Text) <> 0` para evitar la división por cero, mostrando un mensaje de error si ocurre.

Programa 53. Conversor de temperatura (Celsius-Fahrenheit)

Este programa convierte una temperatura de grados Celsius a Fahrenheit.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtCelsius`.
  - `Button`: **Name:** `btnConvertir`, **Text:** "Convertir".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnConvertir_Click(sender As Object, e As EventArgs)
Handles btnConvertir.Click
        Dim c As Double = Val(txtCelsius.Text)
        Dim f As Double = (c * 9 / 5) + 32
        lblResultado.Text = c & " °C = " & f & " °F"
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnConvertir`, el evento `btnConvertir_Click` se ejecuta.
  - `Val(txtCelsius.Text)` obtiene el valor numérico de la temperatura en Celsius.
  - La fórmula `(c * 9 / 5) + 32` se utiliza para convertir Celsius a Fahrenheit.
  - El resultado se muestra en `lblResultado.Text` en un formato legible.

Programa 54. Conversor de divisas

Este programa convierte una cantidad de Euros a Dólares Estadounidenses (USD) utilizando una tasa de cambio fija.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtEuros`.
  - `Button`: **Name:** `btnConvertir`, **Text:** "Convertir a USD".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnConvertir_Click(sender As Object, e As EventArgs)
Handles btnConvertir.Click
        Dim euros As Double = Val(txtEuros.Text)
        Dim tasa As Double = 1.1 ' Ejemplo de tasa EUR a USD
        lblResultado.Text = euros & " € = " & (euros * tasa) & " $"
    End Sub
End Class
```

- **Explicación:**
  - El evento `btnConvertir_Click` se activa al hacer clic en el botón.
  - Se obtiene la cantidad de euros de `txtEuros.Text` y se define una `tasa` de cambio de ejemplo (1.1).
  - Se calcula el equivalente en dólares multiplicando los `euros` por la `tasa`.
  - El resultado se muestra en `lblResultado.Text`.

Programa 55. Bloc de notas básico

Este programa es un editor de texto extremadamente simple, consistiendo únicamente en un `TextBox` multilinea que ocupa todo el formulario.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtContenido`, **Multiline:** `True`, **Dock:** `Fill`.
- **Código VB.NET:**

```
Public Class Form1
    ' Sin eventos, solo un TextBox grande para escribir
End Class
```

- **Explicación:**
  - Este programa no requiere código VB.NET en particular, ya que toda su funcionalidad se basa en las propiedades del `TextBox`.
  - Al establecer `Multiline` a `True`, el `TextBox` puede mostrar varias líneas de texto.
  - Al establecer `Dock` a `Fill`, el `TextBox` se ajusta automáticamente para llenar todo el espacio del formulario, creando una interfaz de bloc de notas básica.

Programa 56. Aplicación que guarda y carga texto desde archivo

Este programa expande la funcionalidad de un bloc de notas básico para permitir guardar el texto en un archivo y cargar texto desde un archivo.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtContenido`, **Multiline:** `True`, **Size:** `300x200`.
  - `Button`: **Name:** `btnGuardar`, **Text:** "Guardar".
  - `Button`: **Name:** `btnCargar`, **Text:** "Cargar".
  - `OpenFileDialog`: **Name:** `OpenFileDialog1`.
  - `SaveFileDialog`: **Name:** `SaveFileDialog1`.
- **Código VB.NET:**

```
Imports System.IO
Public Class Form1
    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click
        If SaveFileDialog1.ShowDialog() = DialogResult.OK Then
            File.WriteAllText(SaveFileDialog1.FileName, txtContenido.Text)
        End If
    End Sub
    Private Sub btnCargar_Click(sender As Object, e As EventArgs) Handles
btnCargar.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            txtContenido.Text = File.ReadAllText(OpenFileDialog1.FileName)
        End If
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.IO` para operaciones de archivo.
  - El evento `btnGuardar_Click` es similar al Programa 50: utiliza `SaveFileDialog1` para obtener una ruta y `File.WriteAllText` para guardar el contenido de `txtContenido` en el archivo.
  - El evento `btnCargar_Click` utiliza `OpenFileDialog1` para que el usuario seleccione un archivo. Si se selecciona, `File.ReadAllText` lee todo el contenido del archivo y lo asigna a `txtContenido.Text`, mostrando el texto en el `TextBox`.

Programa 57. Aplicación que cuenta palabras en un TextBox

Este programa cuenta el número de palabras en un `TextBox` multilinea.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtTexto`, **Multiline:** `True`, **Size:** `300x100`.
  - `Button`: **Name:** `btnContar`, **Text:** "Contar Palabras".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnContar_Click(sender As Object, e As EventArgs) Handles
btnContar.Click
        Dim palabras() As String = txtTexto.Text.Split({" "c, vbCrLf},
StringSplitOptions.RemoveEmptyEntries)
        lblResultado.Text = "Número de palabras: " & palabras.Length
    End Sub
End Class
```

- **Explicación:**
  - Cuando se hace clic en `btnContar`, el evento `btnContar_Click` se ejecuta.
  - `txtTexto.Text.Split({" "c, vbCrLf}, StringSplitOptions.RemoveEmptyEntries)` divide la cadena de texto de `txtTexto` en un array de palabras. Los delimitadores son el espacio y el salto de línea (`vbCrLf`). `StringSplitOptions.RemoveEmptyEntries` asegura que los espacios múltiples o líneas vacías no se cuenten como palabras.
  - `palabras.Length` devuelve la cantidad de elementos en el array (es decir, el número de palabras), que se muestra en `lblResultado.Text`.

Programa 58. Aplicación que muestra tablas de multiplicar

Este programa genera y muestra la tabla de multiplicar de un número ingresado por el usuario en un `ListBox`.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtNumero`.
  - `Button`: **Name:** `btnGenerar`, **Text:** "Generar Tabla".
  - `ListBox`: **Name:** `lstTabla`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnGenerar_Click(sender As Object, e As EventArgs) Handles
btnGenerar.Click
        lstTabla.Items.Clear()
        Dim n As Integer = Val(txtNumero.Text)
        For i As Integer = 1 To 10
            lstTabla.Items.Add(n & " x " & i & " = " & (n * i))
        Next
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnGenerar`, el evento `btnGenerar_Click` se activa.
  - `lstTabla.Items.Clear()` limpia cualquier contenido anterior del `ListBox`.
  - `Val(txtNumero.Text)` obtiene el número del `TextBox`.
  - Un bucle `For` itera del 1 al 10. En cada iteración, se construye una cadena con la operación de multiplicación (`n & " x " & i & " = " & (n * i)`) y se añade como un nuevo elemento a `lstTabla`.

Programa 59. Aplicación que genera contraseñas aleatorias

Este programa genera una contraseña aleatoria de longitud fija (8 caracteres) utilizando una combinación de letras, números y símbolos.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnGenerar`, **Text:** "Generar Contraseña".
  - `Label`: **Name:** `lblPassword`.
- **Código VB.NET:**

```
Imports System.Text
Public Class Form1
    Private Sub btnGenerar_Click(sender As Object, e As EventArgs) Handles
btnGenerar.Click
        Dim chars As String =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%"
        Dim rnd As New Random()
        Dim sb As New StringBuilder()
        For i As Integer = 1 To 8
            sb.Append(chars(rnd.Next(chars.Length)))
        Next
        lblPassword.Text = sb.ToString()
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.Text` para usar la clase `StringBuilder`.
  - Cuando se hace clic en `btnGenerar`, se define una cadena `chars` que contiene todos los caracteres posibles para la contraseña.
  - Se crea una instancia de `Random` (`rnd`) para generar números aleatorios.
  - Un `StringBuilder` (`sb`) se utiliza para construir la contraseña de manera eficiente.
  - Un bucle `For` se ejecuta 8 veces (para una contraseña de 8 caracteres). En cada iteración, `rnd.Next(chars.Length)` selecciona un índice aleatorio dentro de la cadena `chars`, y el carácter en ese índice se añade a `sb`.
  - Finalmente, `lblPassword.Text = sb.ToString()` asigna la contraseña generada al `Label`.

Programa 60. Aplicación que muestra mensajes aleatorios

Este programa muestra un mensaje aleatorio de una lista predefinida cada vez que se presiona un botón.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnMensaje`, **Text:** "Mostrar Mensaje".
  - `Label`: **Name:** `lblMensaje`.
- **Código VB.NET:**

```
Public Class Form1
    Dim mensajes() As String = {"Hola!", "Bienvenido", "VB.NET es genial",
"Sigue aprendiendo", "Suerte con tus proyectos"}
    Private Sub btnMensaje_Click(sender As Object, e As EventArgs) Handles
btnMensaje.Click
        Dim rnd As New Random()
        Dim index As Integer = rnd.Next(mensajes.Length)
        lblMensaje.Text = mensajes(index)
    End Sub
End Class
```

- **Explicación:**
  - Se declara un array de cadenas `mensajes()` a nivel de clase, conteniendo los mensajes posibles.
  - Al hacer clic en `btnMensaje`, el evento `btnMensaje_Click` se ejecuta.
  - Se crea una instancia de `Random` (`rnd`).
  - `rnd.Next(mensajes.Length)` genera un índice aleatorio que está dentro de los límites del array `mensajes`.
  - `lblMensaje.Text = mensajes(index)` selecciona el mensaje en el índice aleatorio y lo muestra en el `Label`.

Programa 61. Aplicación que calcula el IMC

Este programa calcula el Índice de Masa Corporal (IMC) a partir del peso y la altura ingresados por el usuario.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtPeso`, **Placeholder:** "Peso (kg)".
  - `TextBox`: **Name:** `txtAltura`, **Placeholder:** "Altura (m)".
  - `Button`: **Name:** `btnCalcular`, **Text:** "Calcular IMC".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnCalcular_Click(sender As Object, e As EventArgs)
Handles btnCalcular.Click
        Dim peso As Double = Val(txtPeso.Text)
        Dim altura As Double = Val(txtAltura.Text)
        If altura > 0 Then
            Dim imc As Double = peso / (altura * altura)
            lblResultado.Text = "IMC: " & imc.ToString("0.00")
        Else
            lblResultado.Text = "Altura inválida"
        End If
    End Sub
End Class
```

- **Explicación:**
  - El evento `btnCalcular_Click` se activa cuando se presiona el botón "Calcular IMC".
  - Se obtienen los valores de peso y altura de los `TextBox` y se convierten a tipo `Double` usando `Val()`.
  - Se verifica que la `altura` sea mayor que cero para evitar divisiones por cero.
  - El IMC se calcula con la fórmula `peso / (altura * altura)`.
  - El resultado del IMC se muestra en el `lblResultado`, formateado a dos decimales con `ToString("0.00")`. Si la altura es inválida (cero o negativa), se muestra un mensaje de error.

Programa 62. Conversor de unidades de longitud (metros a cm, mm, km)

Este programa convierte una cantidad dada en metros a centímetros, milímetros y kilómetros.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtMetros`.
  - `Button`: **Name:** `btnConvertir`, **Text:** "Convertir".
  - `ListBox`: **Name:** `lstResultados`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnConvertir_Click(sender As Object, e As EventArgs)
Handles btnConvertir.Click
        lstResultados.Items.Clear()
        Dim m As Double = Val(txtMetros.Text)
        lstResultados.Items.Add(m & " m = " & (m * 100) & " cm")
        lstResultados.Items.Add(m & " m = " & (m * 1000) & " mm")
        lstResultados.Items.Add(m & " m = " & (m / 1000) & " km")
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnConvertir`, se limpia `lstResultados` y se obtiene el valor en metros de `txtMetros.Text`.
  - Se realizan las conversiones multiplicando por 100 para centímetros, por 1000 para milímetros, y dividiendo por 1000 para kilómetros.
  - Cada resultado se añade como un nuevo elemento a `lstResultados`.

Programa 63. Conversor de unidades de peso (kg a g, lb)

Este programa convierte una cantidad de kilogramos a gramos y libras.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtKg`.
  - `Button`: **Name:** `btnConvertir`, **Text:** "Convertir".
  - `ListBox`: **Name:** `lstResultados`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnConvertir_Click(sender As Object, e As EventArgs)
Handles btnConvertir.Click
        lstResultados.Items.Clear()
        Dim kg As Double = Val(txtKg.Text)
        lstResultados.Items.Add(kg & " kg = " & (kg * 1000) & " g")
        lstResultados.Items.Add(kg & " kg = " & (kg * 2.20462) & " lb")
    End Sub
End Class
```

- **Explicación:**
  - El evento `btnConvertir_Click` limpia la lista de resultados.
  - Obtiene la cantidad en kilogramos de `txtKg.Text`.
  - Convierte a gramos multiplicando por 1000 y a libras multiplicando por `2.20462`.
  - Los resultados se añaden al `lstResultados`.

Programa 64. Conversor de unidades de tiempo (horas a minutos y segundos)

Este programa convierte una cantidad de horas a minutos y segundos.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtHoras`.
  - `Button`: **Name:** `btnConvertir`, **Text:** "Convertir".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnConvertir_Click(sender As Object, e As EventArgs)
Handles btnConvertir.Click
        Dim h As Double = Val(txtHoras.Text)
        lblResultado.Text = h & " h = " & (h * 60) & " min = " & (h *
3600) & " seg"
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnConvertir`, se obtiene el valor en horas de `txtHoras.Text`.
  - Se calcula el equivalente en minutos (`h * 60`) y en segundos (`h * 3600`).
  - El resultado completo se muestra en `lblResultado.Text`.

Programa 65. Conversor de unidades de velocidad (km/h a m/s)

Este programa convierte una velocidad de kilómetros por hora (km/h) a metros por segundo (m/s).

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtKmH`.
  - `Button`: **Name:** `btnConvertir`, **Text:** "Convertir".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnConvertir_Click(sender As Object, e As EventArgs)
Handles btnConvertir.Click
        Dim kmh As Double = Val(txtKmH.Text)
        Dim ms As Double = kmh / 3.6
        lblResultado.Text = kmh & " km/h = " & ms.ToString("0.00") & "
m/s"
    End Sub
End Class
```

- **Explicación:**
  - El evento `btnConvertir_Click` obtiene el valor en km/h de `txtKmH.Text`.
  - La conversión a m/s se realiza dividiendo por `3.6` (ya que 1 km/h = 1000m / 3600s = 1/3.6 m/s).
  - El resultado se muestra en `lblResultado.Text`, formateado a dos decimales.

Programa 66. Aplicación que guarda configuraciones en My.Settings

Este programa demuestra cómo guardar y cargar una cadena de texto utilizando las configuraciones de la aplicación (`My.Settings`).

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtDato`.
  - `Button`: **Name:** `btnGuardar`, **Text:** "Guardar".
  - `Button`: **Name:** `btnCargar`, **Text:** "Cargar".
  - `Label`: **Name:** `lblResultado`.
- **Configuración adicional:**
  - En `My Project > Settings`, se debe agregar una configuración llamada `MiDato` de tipo `String` y con alcance `User`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click
        My.Settings.MiDato = txtDato.Text
        My.Settings.Save()
    End Sub
    Private Sub btnCargar_Click(sender As Object, e As EventArgs) Handles
btnCargar.Click
        lblResultado.Text = "Dato guardado: " & My.Settings.MiDato
    End Sub
End Class
```

- **Explicación:**
  - El evento `btnGuardar_Click` asigna el texto de `txtDato.Text` a la propiedad `My.Settings.MiDato` y luego llama a `My.Settings.Save()` para persistir el cambio.
  - El evento `btnCargar_Click` recupera el valor de `My.Settings.MiDato` y lo muestra en `lblResultado.Text`. Este valor se mantiene incluso después de cerrar y volver a abrir la aplicación.

Programa 67. Calculadora de área de figuras geométricas (círculo y cuadrado)

Este programa calcula el área de un círculo y un cuadrado basándose en el radio y el lado ingresados, respectivamente.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtRadio` (para círculo).
  - `TextBox`: **Name:** `txtLado` (para cuadrado).
  - `Button`: **Name:** `btnCalcular`, **Text:** "Calcular Áreas".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnCalcular_Click(sender As Object, e As EventArgs)
Handles btnCalcular.Click
        Dim radio As Double = Val(txtRadio.Text)
        Dim lado As Double = Val(txtLado.Text)
        Dim areaCirculo As Double = Math.PI * radio ^ 2
        Dim areaCuadrado As Double = lado * lado
        lblResultado.Text = "Área círculo: " &
areaCirculo.ToString("0.00") & vbCrLf & _
                            "Área cuadrado: " & areaCuadrado
    End Sub
End Class
```

- **Explicación:**
  - Al presionar `btnCalcular`, se obtienen los valores de `txtRadio.Text` y `txtLado.Text`.
  - El área del círculo se calcula usando `Math.PI * radio ^ 2`.
  - El área del cuadrado se calcula como `lado * lado`.
  - Ambos resultados se muestran en `lblResultado.Text`, con el área del círculo formateada a dos decimales y un salto de línea (`vbCrLf`) entre ellas.

Programa 68. Calculadora de perímetro de figuras geométricas (círculo y cuadrado)

Este programa calcula el perímetro de un círculo y un cuadrado, similar al anterior, pero para perímetros.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtRadio`.
  - `TextBox`: **Name:** `txtLado`.
  - `Button`: **Name:** `btnCalcular`, **Text:** "Calcular Perímetros".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnCalcular_Click(sender As Object, e As EventArgs)
Handles btnCalcular.Click
        Dim radio As Double = Val(txtRadio.Text)
        Dim lado As Double = Val(txtLado.Text)
        Dim perCirculo As Double = 2 * Math.PI * radio
        Dim perCuadrado As Double = 4 * lado
        lblResultado.Text = "Perímetro círculo: " &
perCirculo.ToString("0.00") & vbCrLf & _
                            "Perímetro cuadrado: " & perCuadrado
    End Sub
End Class
```

- **Explicación:**
  - El evento `btnCalcular_Click` obtiene el radio y el lado.
  - El perímetro del círculo se calcula como `2 * Math.PI * radio`.
  - El perímetro del cuadrado se calcula como `4 * lado`.
  - Los resultados se muestran en `lblResultado.Text`, con el perímetro del círculo formateado a dos decimales y un salto de línea.

Programa 69. Juego del número secreto

Este programa implementa un juego simple donde el usuario debe adivinar un número secreto generado aleatoriamente entre 1 y 100.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtIntento`.
  - `Button`: **Name:** `btnProbar`, **Text:** "Probar".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Dim secreto As Integer
    Dim rnd As New Random()
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        secreto = rnd.Next(1, 101) ' Número entre 1 y 100
    End Sub
    Private Sub btnProbar_Click(sender As Object, e As EventArgs) Handles
btnProbar.Click
        Dim intento As Integer = Val(txtIntento.Text)
        If intento = secreto Then
            lblResultado.Text = "¡Correcto!"
        ElseIf intento < secreto Then
            lblResultado.Text = "El número es mayor"
        Else
            lblResultado.Text = "El número es menor"
        End If
    End Sub
End Class
```

- **Explicación:**
  - Se declaran a nivel de clase un `Integer` `secreto` y un objeto `Random`.
  - En `Form1_Load`, se genera un número aleatorio entre 1 y 100 (inclusive) y se asigna a `secreto`.
  - El evento `btnProbar_Click` obtiene el intento del usuario.
  - Compara el `intento` con el `secreto` y actualiza `lblResultado.Text` para indicar si es "¡Correcto!", "El número es mayor" o "El número es menor".

Programa 70. Juego de adivinar palabra

Este programa es un juego simple de adivinanza de palabras donde el usuario introduce letras para descubrir una palabra oculta.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblPista`, **Text:** "Palabra: _ _ _ _".
  - `TextBox`: **Name:** `txtLetra`.
  - `Button`: **Name:** `btnProbar`, **Text:** "Probar Letra".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Dim palabra As String = "GATO"
    Dim adivinada As String = "____"
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        lblPista.Text = "Palabra: " & adivinada
    End Sub
    Private Sub btnProbar_Click(sender As Object, e As EventArgs) Handles
btnProbar.Click
        Dim letra As String = txtLetra.Text.ToUpper()
        If letra.Length = 1 Then
            Dim nueva As Char() = adivinada.ToCharArray()
            For i As Integer = 0 To palabra.Length - 1
                If palabra(i) = letra(0) Then
                    nueva(i) = letra(0)
                End If
            Next
            adivinada = New String(nueva)
            lblPista.Text = "Palabra: " & adivinada
            If adivinada = palabra Then
                lblResultado.Text = "¡Ganaste!"
            End If
        End If
        txtLetra.Clear()
    End Sub
End Class
```

- **Explicación:**
  - `palabra` es la palabra a adivinar, y `adivinada` es la representación oculta (guiones bajos).
  - En `Form1_Load`, `lblPista` muestra la palabra oculta inicial.
  - En `btnProbar_Click`, se obtiene la `letra` ingresada y se convierte a mayúscula.
  - Si se ingresa solo una letra, se itera sobre la `palabra` para ver si la letra coincide con alguna posición. Si hay coincidencia, se revela la letra en la `adivinada`.
  - `lblPista` se actualiza, y si la palabra `adivinada` es igual a la `palabra` original, el usuario gana. El `TextBox` `txtLetra` se limpia para la siguiente entrada.

Programa 71. Juego de piedra, papel o tijera

Este programa simula el juego de piedra, papel o tijera contra la computadora.

- **Controles necesarios en el formulario:**
  - `ComboBox`: **Name:** `cmbJugador`, **Items:** {"Piedra", "Papel", "Tijera"}.
  - `Button`: **Name:** `btnJugar`, **Text:** "Jugar".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Dim opciones() As String = {"Piedra", "Papel", "Tijera"}
    Dim rnd As New Random()
    Private Sub btnJugar_Click(sender As Object, e As EventArgs) Handles
btnJugar.Click
        Dim jugador As String = cmbJugador.Text
        Dim pc As String = opciones(rnd.Next(opciones.Length))
        If jugador = "" Then
            lblResultado.Text = "Elige una opción"
            Exit Sub
        End If
        If jugador = pc Then
            lblResultado.Text = "Empate, PC eligió " & pc
        ElseIf (jugador = "Piedra" And pc = "Tijera") Or _
               (jugador = "Papel" And pc = "Piedra") Or _
               (jugador = "Tijera" And pc = "Papel") Then
            lblResultado.Text = "Ganaste, PC eligió " & pc
        Else
            lblResultado.Text = "Perdiste, PC eligió " & pc
        End If
    End Sub
End Class
```

- **Explicación:**
  - Un array `opciones` guarda las posibles jugadas, y un `Random` genera la jugada del PC.
  - En `btnJugar_Click`, se obtiene la elección del `jugador` del `ComboBox` y la del `pc` aleatoriamente.
  - Se valida que el `jugador` haya elegido una opción.
  - La lógica del juego compara las elecciones para determinar si es "Empate", "Ganaste" o "Perdiste", y muestra el resultado junto con la elección del PC en `lblResultado`.

Programa 72. Cronómetro con Start, Stop y Reset

Este programa implementa un cronómetro básico con funcionalidades de inicio, parada y reinicio.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblTiempo`, **Text:** "00:00:00".
  - `Button`: **Name:** `btnStart`, **Text:** "Start".
  - `Button`: **Name:** `btnStop`, **Text:** "Stop".
  - `Button`: **Name:** `btnReset`, **Text:** "Reset".
  - `Timer`: **Name:** `Timer1`, **Interval:** `1000`.
- **Código VB.NET:**

```
Public Class Form1
    Dim segundos As Integer = 0
    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
        segundos += 1
        lblTiempo.Text =
TimeSpan.FromSeconds(segundos).ToString("hh\:mm\:ss")
    End Sub
    Private Sub btnStart_Click(sender As Object, e As EventArgs) Handles
btnStart.Click
        Timer1.Start()
    End Sub
    Private Sub btnStop_Click(sender As Object, e As EventArgs) Handles
btnStop.Click
        Timer1.Stop()
    End Sub
    Private Sub btnReset_Click(sender As Object, e As EventArgs) Handles
btnReset.Click
        Timer1.Stop()
        segundos = 0
        lblTiempo.Text = "00:00:00"
    End Sub
End Class
```

- **Explicación:**
  - La variable `segundos` a nivel de clase almacena el tiempo transcurrido.
  - `Timer1_Tick` se dispara cada segundo (`Interval: 1000`), incrementa `segundos` y actualiza `lblTiempo` con el formato `hh:mm:ss` usando `TimeSpan.FromSeconds`.
  - `btnStart_Click` inicia el `Timer1`.
  - `btnStop_Click` detiene el `Timer1`.
  - `btnReset_Click` detiene el temporizador, reinicia `segundos` a 0 y restablece `lblTiempo` a "00:00:00".

Programa 73. Temporizador con alarma

Este programa permite al usuario establecer un temporizador que, al llegar a cero, muestra una alarma.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtSegundos`, **Placeholder:** "Segundos".
  - `Button`: **Name:** `btnIniciar`, **Text:** "Iniciar".
  - `Label`: **Name:** `lblTiempo`.
  - `Timer`: **Name:** `Timer1`, **Interval:** `1000`.
- **Código VB.NET:**

```
Public Class Form1
    Dim tiempo As Integer
    Private Sub btnIniciar_Click(sender As Object, e As EventArgs) Handles
btnIniciar.Click
        tiempo = Val(txtSegundos.Text)
        Timer1.Start()
    End Sub
    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
        If tiempo > 0 Then
            tiempo -= 1
            lblTiempo.Text = tiempo & " s"
        Else
            Timer1.Stop()
            MessageBox.Show("¡Tiempo terminado!")
        End If
    End Sub
End Class
```

- **Explicación:**
  - La variable `tiempo` guarda los segundos restantes.
  - En `btnIniciar_Click`, se obtiene el valor de `txtSegundos.Text` y se inicia `Timer1`.
  - Cada segundo (`Timer1_Tick`), `tiempo` se decrementa y `lblTiempo` se actualiza.
  - Cuando `tiempo` llega a cero, el `Timer1` se detiene y se muestra un `MessageBox` de alarma.

Programa 74. Editor de listas (agregar, modificar y borrar)

Este programa permite agregar, modificar y borrar elementos en un `ListBox`.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtElemento`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar".
  - `Button`: **Name:** `btnModificar`, **Text:** "Modificar".
  - `Button`: **Name:** `btnBorrar`, **Text:** "Borrar".
  - `ListBox`: **Name:** `lstElementos`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
        lstElementos.Items.Add(txtElemento.Text)
        txtElemento.Clear()
    End Sub
    Private Sub btnModificar_Click(sender As Object, e As EventArgs)
Handles btnModificar.Click
        If lstElementos.SelectedIndex >= 0 Then
            lstElementos.Items(lstElementos.SelectedIndex) =
txtElemento.Text
        End If
    End Sub
    Private Sub btnBorrar_Click(sender As Object, e As EventArgs) Handles
btnBorrar.Click
        If lstElementos.SelectedIndex >= 0 Then
            lstElementos.Items.RemoveAt(lstElementos.SelectedIndex)
        End If
    End Sub
End Class
```

- **Explicación:**
  - `btnAgregar_Click` añade el texto de `txtElemento.Text` a `lstElementos` y luego limpia el `TextBox`.
  - `btnModificar_Click` verifica si hay un elemento seleccionado (`lstElementos.SelectedIndex >= 0`). Si es así, reemplaza el elemento seleccionado con el texto actual de `txtElemento.Text`.
  - `btnBorrar_Click` también verifica una selección y, si existe, elimina el elemento en esa posición de `lstElementos`.

Programa 75. Editor de texto con guardar y abrir archivo

Este programa es un editor de texto simple que permite abrir un archivo de texto existente y guardar el contenido en un nuevo archivo.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtContenido`, **Multiline:** `True`, **Dock:** `Fill`.
  - `MenuStrip`: **Items:** {"Archivo → Abrir", "Archivo → Guardar"}.
  - `OpenFileDialog`: **Name:** `OpenFileDialog1`.
  - `SaveFileDialog`: **Name:** `SaveFileDialog1`.
- **Código VB.NET:**

```
Imports System.IO
Public Class Form1
    Private Sub AbrirToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles AbrirToolStripMenuItem.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            txtContenido.Text = File.ReadAllText(OpenFileDialog1.FileName)
        End If
    End Sub
    Private Sub GuardarToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles GuardarToolStripMenuItem.Click
        If SaveFileDialog1.ShowDialog() = DialogResult.OK Then
            File.WriteAllText(SaveFileDialog1.FileName, txtContenido.Text)
        End If
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.IO` para las operaciones de archivo.
  - El evento `AbrirToolStripMenuItem_Click` utiliza `OpenFileDialog1` para que el usuario seleccione un archivo. Si el diálogo retorna `OK`, `File.ReadAllText` lee el contenido del archivo y lo carga en `txtContenido.Text`.
  - El evento `GuardarToolStripMenuItem_Click` utiliza `SaveFileDialog1` para que el usuario elija dónde guardar el archivo. Si el diálogo retorna `OK`, `File.WriteAllText` guarda el contenido de `txtContenido.Text` en el archivo especificado.

Programa 76. Aplicación con varios formularios enlazados

Este programa muestra cómo un formulario (`Form1`) puede abrir otro formulario (`Form2`).

- **Controles necesarios en Form1:**
  - `Button`: **Name:** `btnAbrir`, **Text:** "Abrir Form2".
- **Controles necesarios en Form2:**
  - `Form2` debe crearse como un nuevo formulario vacío en el proyecto, con un `Label` opcional.
- **Código VB.NET:**

```
' Form1
Public Class Form1
    Private Sub btnAbrir_Click(sender As Object, e As EventArgs) Handles
btnAbrir.Click
        Dim f2 As New Form2
        f2.Show()
    End Sub
End Class
' Form2
Public Class Form2
    ' Puede estar vacío o contener un Label como sugerencia en la fuente
End Class
```

- **Explicación:**
  - En `Form1`, al hacer clic en `btnAbrir`, se crea una nueva instancia de `Form2` (`Dim f2 As New Form2`) y luego se muestra con `f2.Show()`. Esto abre una segunda ventana.

Programa 77. Agenda que guarda datos en archivo de texto

Este programa es una agenda simple que permite guardar contactos (nombre y teléfono) en un archivo de texto plano.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtNombre`.
  - `TextBox`: **Name:** `txtTelefono`.
  - `Button`: **Name:** `btnGuardar`, **Text:** "Guardar".
  - `ListBox`: **Name:** `lstContactos`.
- **Código VB.NET:**

```
Imports System.IO
Public Class Form1
    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click
        Dim contacto As String = txtNombre.Text & " - " & txtTelefono.Text
        File.AppendAllText("agenda.txt", contacto & Environment.NewLine)
        lstContactos.Items.Add(contacto)
        txtNombre.Clear()
        txtTelefono.Clear()
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.IO`.
  - En `btnGuardar_Click`, se concatena el nombre y el teléfono para formar una cadena `contacto`.
  - `File.AppendAllText("agenda.txt", contacto & Environment.NewLine)` añade esta cadena, seguida de un salto de línea, al final del archivo "agenda.txt".
  - El contacto también se añade a `lstContactos` y los `TextBox` se limpian.

Programa 78. Agenda que guarda datos en archivo CSV

Este programa guarda contactos en un archivo de formato CSV (Comma Separated Values), que es compatible con hojas de cálculo.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtNombre`.
  - `TextBox`: **Name:** `txtTelefono`.
  - `Button`: **Name:** `btnGuardar`, **Text:** "Guardar".
  - `ListBox`: **Name:** `lstContactos`.
- **Código VB.NET:**

```
Imports System.IO
Public Class Form1
    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click
        Dim contacto As String = txtNombre.Text & "," & txtTelefono.Text
        File.AppendAllText("agenda.csv", contacto & Environment.NewLine)
        lstContactos.Items.Add(contacto)
        txtNombre.Clear()
        txtTelefono.Clear()
    End Sub
End Class
```

- **Explicación:**
  - Similar al Programa 77, pero el `contacto` se formatea con una coma como separador (`txtNombre.Text & "," & txtTelefono.Text`).
  - `File.AppendAllText("agenda.csv", ...)` guarda la línea en un archivo llamado "agenda.csv".

Programa 79. Agenda que guarda datos en archivo XML

Este programa guarda contactos en un archivo XML, utilizando la estructura jerárquica de este formato.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtNombre`.
  - `TextBox`: **Name:** `txtTelefono`.
  - `Button`: **Name:** `btnGuardar`, **Text:** "Guardar".
  - `ListBox`: **Name:** `lstContactos`.
- **Código VB.NET:**

```
Imports System.Xml
Public Class Form1
    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click
        Dim doc As New XmlDocument()
        Dim archivo As String = "agenda.xml"
        If IO.File.Exists(archivo) Then
            doc.Load(archivo)
        Else
            Dim root As XmlElement = doc.CreateElement("Contactos")
            doc.AppendChild(root)
        End If
        Dim contacto As XmlElement = doc.CreateElement("Contacto")
        contacto.SetAttribute("Nombre", txtNombre.Text)
        contacto.SetAttribute("Telefono", txtTelefono.Text)
        doc.DocumentElement.AppendChild(contacto)
        doc.Save(archivo)
        lstContactos.Items.Add(txtNombre.Text & " - " & txtTelefono.Text)
        txtNombre.Clear()
        txtTelefono.Clear()
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.Xml`.
  - En `btnGuardar_Click`, se crea un `XmlDocument`. Si el archivo "agenda.xml" existe, se carga; de lo contrario, se crea un nuevo elemento raíz llamado "Contactos".
  - Se crea un elemento "Contacto" para cada nueva entrada.
  - El nombre y el teléfono se guardan como atributos de este elemento "Contacto".
  - El nuevo elemento "Contacto" se añade como hijo del elemento raíz (o del documento si es la primera vez).
  - `doc.Save(archivo)` guarda el documento XML. El contacto también se añade a `lstContactos` y los `TextBox` se limpian.

Programa 80. Agenda que guarda datos en base de datos Access

Este programa demuestra cómo guardar contactos en una base de datos Microsoft Access.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtNombre`.
  - `TextBox`: **Name:** `txtTelefono`.
  - `Button`: **Name:** `btnGuardar`, **Text:** "Guardar".
  - `ListBox`: **Name:** `lstContactos`.
- **Requisito adicional:**
  - Requiere tener una base de datos Access llamada `agenda.accdb` con una tabla `Contactos` que tenga campos `Nombre` y `Telefono`.
- **Código VB.NET:**

```
Imports System.Data.OleDb
Public Class Form1
    Dim conexion As New
OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=agenda.accdb;")
    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click
        Dim cmd As New OleDbCommand("INSERT INTO Contactos (Nombre,
Telefono) VALUES (@n, @t)", conexion)
        cmd.Parameters.AddWithValue("@n", txtNombre.Text)
        cmd.Parameters.AddWithValue("@t", txtTelefono.Text)
        conexion.Open()
        cmd.ExecuteNonQuery()
        conexion.Close()
        lstContactos.Items.Add(txtNombre.Text & " - " & txtTelefono.Text)
        txtNombre.Clear()
        txtTelefono.Clear()
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.Data.OleDb`.
  - Se declara una variable `conexion` de tipo `OleDbConnection` configurada para conectarse a `agenda.accdb`.
  - En `btnGuardar_Click`, se crea un `OleDbCommand` con una consulta `INSERT` parametrizada.
  - Se añaden los valores de `txtNombre.Text` y `txtTelefono.Text` como parámetros a la consulta.
  - Se abre la `conexion`, se ejecuta la consulta (`cmd.ExecuteNonQuery()`) y se cierra la `conexion`.
  - Finalmente, el contacto se añade a `lstContactos` y los `TextBox` se limpian.

Programa 81. Formulario con gráficos simples

Este programa demuestra cómo dibujar formas geométricas básicas (líneas, rectángulos, elipses) directamente sobre un `PictureBox` utilizando la clase `Graphics`.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnDibujar`, **Text:** "Dibujar".
  - `PictureBox`: **Name:** `picGrafico`, **Size:** `300x200`, **BorderStyle:** `FixedSingle`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnDibujar_Click(sender As Object, e As EventArgs) Handles
btnDibujar.Click
        Dim g As Graphics = picGrafico.CreateGraphics()
        g.Clear(Color.White)
        g.DrawLine(Pens.Blue, 10, 10, 200, 100)
        g.DrawRectangle(Pens.Red, 50, 50, 100, 50)
        g.DrawEllipse(Pens.Green, 100, 30, 80, 40)
    End Sub
End Class
```

- **Explicación:**
  - En `btnDibujar_Click`, se obtiene un objeto `Graphics` a partir del `PictureBox` `picGrafico`.
  - `g.Clear(Color.White)` limpia el `PictureBox` con color blanco.
  - `g.DrawLine()` dibuja una línea azul.
  - `g.DrawRectangle()` dibuja un rectángulo rojo.
  - `g.DrawEllipse()` dibuja una elipse verde. Los parámetros especifican el lápiz (color), y las coordenadas/dimensiones.

Programa 82. Formulario con impresión de texto

Este programa permite imprimir el contenido de un `TextBox` utilizando los controles `PrintDocument` y `PrintDialog`.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtTexto`, **Multiline:** `True`, **Size:** `300x150`.
  - `Button`: **Name:** `btnImprimir`, **Text:** "Imprimir".
  - `PrintDocument`: **Name:** `PrintDocument1`.
  - `PrintDialog`: **Name:** `PrintDialog1`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnImprimir_Click(sender As Object, e As EventArgs)
Handles btnImprimir.Click
        If PrintDialog1.ShowDialog() = DialogResult.OK Then
            PrintDocument1.Print()
        End If
    End Sub
    Private Sub PrintDocument1_PrintPage(sender As Object, e As
Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage
        e.Graphics.DrawString(txtTexto.Text, New Font("Arial", 12),
Brushes.Black, 50, 50)
    End Sub
End Class
```

- **Explicación:**
  - El evento `btnImprimir_Click` muestra el `PrintDialog1`. Si el usuario confirma la impresión, se llama a `PrintDocument1.Print()`.
  - Cuando se llama a `PrintDocument1.Print()`, el evento `PrintDocument1_PrintPage` se dispara para renderizar cada página a imprimir.
  - Dentro de `PrintDocument1_PrintPage`, `e.Graphics.DrawString()` dibuja el texto de `txtTexto.Text` en la página, utilizando una fuente Arial de tamaño 12, color negro y en la posición (50, 50).

Programa 83. Calculadora de ecuaciones de segundo grado

Este programa resuelve ecuaciones de segundo grado ($Ax^2 + Bx + C = 0$) y muestra sus soluciones reales.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtA`.
  - `TextBox`: **Name:** `txtB`.
  - `TextBox`: **Name:** `txtC`.
  - `Button`: **Name:** `btnCalcular`, **Text:** "Calcular".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnCalcular_Click(sender As Object, e As EventArgs)
Handles btnCalcular.Click
        Dim a As Double = Val(txtA.Text)
        Dim b As Double = Val(txtB.Text)
        Dim c As Double = Val(txtC.Text)
        Dim d As Double = b ^ 2 - 4 * a * c ' Discriminante
        If d < 0 Then
            lblResultado.Text = "No hay soluciones reales"
        Else
            Dim x1 As Double = (-b + Math.Sqrt(d)) / (2 * a)
            Dim x2 As Double = (-b - Math.Sqrt(d)) / (2 * a)
            lblResultado.Text = "x1=" & x1 & ", x2=" & x2
        End If
    End Sub
End Class
```

- **Explicación:**
  - En `btnCalcular_Click`, se obtienen los coeficientes A, B y C de los `TextBox`.
  - Se calcula el discriminante `d = b^2 - 4ac`.
  - Si `d < 0`, no hay soluciones reales y se informa en `lblResultado`.
  - Si `d >= 0`, se calculan las dos raíces `x1` y `x2` usando la fórmula general (`(-b ± √d) / (2a)`) y se muestran en `lblResultado`.

Programa 84. Calculadora de interés simple

Este programa calcula el interés simple generado por un capital, una tasa de interés y un período de tiempo.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtCapital`.
  - `TextBox`: **Name:** `txtInteres` (porcentaje).
  - `TextBox`: **Name:** `txtTiempo` (años).
  - `Button`: **Name:** `btnCalcular`, **Text:** "Calcular".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnCalcular_Click(sender As Object, e As EventArgs)
Handles btnCalcular.Click
        Dim C As Double = Val(txtCapital.Text)
        Dim i As Double = Val(txtInteres.Text) / 100
        Dim t As Double = Val(txtTiempo.Text)
        Dim interés As Double = C * i * t
        lblResultado.Text = "Interés simple: " & interés
    End Sub
End Class
```

- **Explicación:**
  - En `btnCalcular_Click`, se obtienen el capital (C), el interés (i, convertido a decimal) y el tiempo (t) de los `TextBox`.
  - La fórmula `C * i * t` se utiliza para calcular el interés simple.
  - El resultado se muestra en `lblResultado.Text`.

Programa 85. Calculadora de interés compuesto

Este programa calcula el monto final de una inversión con interés compuesto, basándose en el capital, la tasa de interés y el tiempo.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtCapital`.
  - `TextBox`: **Name:** `txtInteres` (porcentaje).
  - `TextBox`: **Name:** `txtTiempo` (años).
  - `Button`: **Name:** `btnCalcular`, **Text:** "Calcular".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnCalcular_Click(sender As Object, e As EventArgs)
Handles btnCalcular.Click
        Dim C As Double = Val(txtCapital.Text)
        Dim i As Double = Val(txtInteres.Text) / 100
        Dim t As Double = Val(txtTiempo.Text)
        Dim monto As Double = C * (1 + i) ^ t
        lblResultado.Text = "Monto con interés compuesto: " &
monto.ToString("0.00")
    End Sub
End Class
```

- **Explicación:**
  - El evento `btnCalcular_Click` obtiene el capital (C), el interés (i, convertido a decimal) y el tiempo (t).
  - La fórmula `C * (1 + i) ^ t` se aplica para calcular el monto final con interés compuesto.
  - El resultado se muestra en `lblResultado.Text`, formateado a dos decimales.

Programa 86. Conversor de números decimales a binarios

Este programa convierte un número decimal (base 10) a su representación binaria (base 2).

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtDecimal`.
  - `Button`: **Name:** `btnConvertir`, **Text:** "Convertir".
  - `Label`: **Name:** `lblBinario`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnConvertir_Click(sender As Object, e As EventArgs)
Handles btnConvertir.Click
        Dim n As Integer = Val(txtDecimal.Text)
        lblBinario.Text = "Binario: " & Convert.ToString(n, 2)
    End Sub
End Class
```

- **Explicación:**
  - En `btnConvertir_Click`, se obtiene el número decimal de `txtDecimal.Text`.
  - `Convert.ToString(n, 2)` es un método integrado de .NET que convierte un número entero a su representación en la base especificada (en este caso, base 2 para binario).
  - El resultado binario se muestra en `lblBinario.Text`.

Programa 87. Conversor de binarios a decimales

Este programa convierte un número binario (base 2) a su equivalente decimal (base 10).

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtBinario`.
  - `Button`: **Name:** `btnConvertir`, **Text:** "Convertir".
  - `Label`: **Name:** `lblDecimal`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnConvertir_Click(sender As Object, e As EventArgs)
Handles btnConvertir.Click
        Dim bin As String = txtBinario.Text
        Try
            lblDecimal.Text = "Decimal: " & Convert.ToInt32(bin, 2)
        Catch ex As Exception
            lblDecimal.Text = "Número binario inválido"
        End Try
    End Sub
End Class
```

- **Explicación:**
  - En `btnConvertir_Click`, se obtiene la cadena binaria de `txtBinario.Text`.
  - `Convert.ToInt32(bin, 2)` es un método que convierte una cadena en una base especificada (binario, base 2) a un entero decimal de 32 bits.
  - Un bloque `Try-Catch` maneja el caso en que la cadena no sea un número binario válido, mostrando un mensaje de error.

Programa 88. Conversor de números romanos a enteros

Este programa convierte un número romano ingresado por el usuario a su equivalente entero.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtRomano`.
  - `Button`: **Name:** `btnConvertir`, **Text:** "Convertir".
  - `Label`: **Name:** `lblEntero`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnConvertir_Click(sender As Object, e As EventArgs)
Handles btnConvertir.Click
        Dim roman As String = txtRomano.Text.ToUpper()
        Dim dic As New Dictionary(Of Char, Integer) From { _
            {"I"c, 1}, {"V"c, 5}, {"X"c, 10}, {"L"c, 50}, _
            {"C"c, 100}, {"D"c, 500}, {"M"c, 1000} _
        }
        Dim total As Integer = 0
        Dim prev As Integer = 0
        For i As Integer = roman.Length - 1 To 0 Step -1
            Dim val As Integer = dic(roman(i))
            If val < prev Then
                total -= val
            Else
                total += val
            End If
            prev = val
        Next
        lblEntero.Text = "Entero: " & total
    End Sub
End Class
```

- **Explicación:**
  - En `btnConvertir_Click`, se convierte el texto a mayúsculas.
  - Un `Dictionary` (`dic`) almacena el valor entero de cada símbolo romano.
  - El algoritmo itera sobre la cadena romana desde el final hacia el principio. Compara el valor del símbolo actual con el `prev` (símbolo anterior). Si el valor actual es menor que el anterior (ej. IV = 5-1), se resta; de lo contrario, se suma.
  - El `total` acumulado se muestra en `lblEntero.Text`.

Programa 89. Conversor de horas a minutos y segundos

Este programa convierte un número de horas a su equivalente en minutos y segundos.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtHoras`.
  - `Button`: **Name:** `btnConvertir`, **Text:** "Convertir".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnConvertir_Click(sender As Object, e As EventArgs)
Handles btnConvertir.Click
        Dim h As Double = Val(txtHoras.Text)
        lblResultado.Text = h & " horas = " & (h * 60) & " minutos = " &
(h * 3600) & " segundos"
    End Sub
End Class
```

- **Explicación:**
  - En `btnConvertir_Click`, se obtiene el valor de horas de `txtHoras.Text`.
  - Se calculan los minutos multiplicando las horas por 60, y los segundos por 3600.
  - El resultado completo se muestra en `lblResultado.Text`.

Programa 90. Conversor de kilogramos a libras

Este programa convierte una cantidad de kilogramos a libras.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtKg`.
  - `Button`: **Name:** `btnConvertir`, **Text:** "Convertir".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnConvertir_Click(sender As Object, e As EventArgs)
Handles btnConvertir.Click
        Dim kg As Double = Val(txtKg.Text)
        Dim lb As Double = kg * 2.20462
        lblResultado.Text = kg & " kg = " & lb.ToString("0.00") & " lb"
    End Sub
End Class
```

- **Explicación:**
  - En `btnConvertir_Click`, se obtiene la cantidad en kilogramos de `txtKg.Text`.
  - Se multiplica por el factor de conversión `2.20462` para obtener las libras.
  - El resultado se muestra en `lblResultado.Text`, formateado a dos decimales.

Programa 91. Conversor de metros a pies

Este programa convierte una longitud dada en metros a su equivalente en pies.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtMetros`.
  - `Button`: **Name:** `btnConvertir`, **Text:** "Convertir".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnConvertir_Click(sender As Object, e As EventArgs)
Handles btnConvertir.Click
        Dim metros As Double = Val(txtMetros.Text)
        Dim pies As Double = metros * 3.28084
        lblResultado.Text = metros & " m = " & pies.ToString("0.00") & "
ft"
    End Sub
End Class
```

- **Explicación:**
  - El evento `btnConvertir_Click` se activa cuando el usuario pulsa el botón "Convertir".
  - Se obtiene el valor numérico de los metros desde `txtMetros.Text` y se almacena en la variable `metros` de tipo `Double`.
  - La conversión se realiza multiplicando `metros` por el factor de conversión `3.28084` (el número de pies en un metro), y el resultado se guarda en `pies`.
  - Finalmente, el `lblResultado.Text` muestra la longitud original en metros y su equivalente en pies, formateado a dos decimales con `ToString("0.00")`.

Programa 92. Conversor de Celsius a Kelvin

Este programa convierte una temperatura de grados Celsius a Kelvin.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtCelsius`.
  - `Button`: **Name:** `btnConvertir`, **Text:** "Convertir".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnConvertir_Click(sender As Object, e As EventArgs)
Handles btnConvertir.Click
        Dim c As Double = Val(txtCelsius.Text)
        Dim k As Double = c + 273.15
        lblResultado.Text = c & " °C = " & k.ToString("0.00") & " K"
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnConvertir`, se ejecuta el código dentro de `btnConvertir_Click`.
  - El valor de la temperatura en Celsius se extrae de `txtCelsius.Text` y se convierte a `Double`.
  - Para convertir a Kelvin, se suma `273.15` al valor en Celsius, obteniendo `k`.
  - El resultado se presenta en `lblResultado.Text`, mostrando tanto la temperatura original como la convertida, con el valor en Kelvin formateado a dos decimales.

Programa 93. Conversor de binario a hexadecimal

Este programa convierte un número binario a su representación hexadecimal.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtBinario`.
  - `Button`: **Name:** `btnConvertir`, **Text:** "Convertir".
  - `Label`: **Name:** `lblHex`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnConvertir_Click(sender As Object, e As EventArgs)
Handles btnConvertir.Click
        Try
            Dim n As Integer = Convert.ToInt32(txtBinario.Text, 2)
            lblHex.Text = "Hexadecimal: " & Hex(n)
        Catch ex As Exception
            lblHex.Text = "Número binario inválido"
        End Try
    End Sub
End Class
```

- **Explicación:**
  - El evento `btnConvertir_Click` maneja la conversión.
  - Utiliza un bloque `Try-Catch` para gestionar posibles errores si el texto ingresado no es un binario válido.
  - `Convert.ToInt32(txtBinario.Text, 2)` convierte la cadena binaria (`txtBinario.Text`) a un número entero decimal, especificando que la base de origen es 2.
  - Luego, la función `Hex(n)` convierte ese número entero decimal a su representación hexadecimal, la cual se muestra en `lblHex.Text`. Si hay un error, `lblHex.Text` indicará que el número es inválido.

Programa 94. Conversor de hexadecimal a decimal

Este programa convierte un número hexadecimal a su equivalente decimal.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtHex`.
  - `Button`: **Name:** `btnConvertir`, **Text:** "Convertir".
  - `Label`: **Name:** `lblDecimal`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnConvertir_Click(sender As Object, e As EventArgs)
Handles btnConvertir.Click
        Try
            Dim n As Integer = Convert.ToInt32(txtHex.Text, 16)
            lblDecimal.Text = "Decimal: " & n
        Catch ex As Exception
            lblDecimal.Text = "Número hexadecimal inválido"
        End Try
    End Sub
End Class
```

- **Explicación:**
  - Al presionar `btnConvertir`, el código en `btnConvertir_Click` se ejecuta.
  - Un bloque `Try-Catch` se utiliza para capturar errores de formato.
  - `Convert.ToInt32(txtHex.Text, 16)` toma la cadena hexadecimal de `txtHex.Text` y la convierte a un número entero decimal, indicando que la base de origen es 16.
  - El número decimal resultante se muestra en `lblDecimal.Text`. Si el formato es incorrecto, se notifica un error.

Programa 95. Juego de memoria con botones

Este juego de memoria simple involucra cuatro botones que cambian de color, y el usuario debe recordar y reproducir una secuencia de colores.

- **Controles necesarios en el formulario:**
    - `4 Buttons`: **Name:** `btn1`, `btn2`, `btn3`, `btn4`. (Se sugiere asignar colores aleatorios en `Form_Load` si se desea una mayor interactividad).
- **Código VB.NET:**

```
Public Class Form1
    Dim colores() As Color = {Color.Red, Color.Green, Color.Blue,
Color.Yellow}
    Dim secuencia() As Integer = {0, 1, 2, 3}
    Dim indice As Integer = 0
    Private Sub btn_Click(sender As Object, e As EventArgs) Handles
btn1.Click, btn2.Click, btn3.Click, btn4.Click
        Dim btn As Button = CType(sender, Button)
        If btn.BackColor = colores(secuencia(indice)) Then
            indice += 1
            If indice = secuencia.Length Then
                MessageBox.Show("¡Ganaste!")
                indice = 0
            End If
        Else
            MessageBox.Show("Error")
            indice = 0
        End If
    End Sub
End Class
```

- **Explicación:**
    - Se definen un array `colores` con los colores posibles y un array `secuencia` con el orden esperado de los colores (índices del array `colores`). `indice` lleva la cuenta de la posición actual en la secuencia que el jugador debe adivinar.
    - El evento `btn_Click` se maneja para todos los botones. Cuando se hace clic en un botón, se convierte `sender` a tipo `Button`.
    - Se compara el `BackColor` del botón clicado con el color esperado en la `secuencia` actual.
    - Si el color es correcto, `indice` se incrementa. Si `indice` alcanza la longitud total de la `secuencia`, el jugador "gana" y `indice` se reinicia.
    - Si el color es incorrecto, se muestra un mensaje de "Error" y `indice` también se reinicia.
    - **Nota:** Para que el juego sea dinámico, la `secuencia` y los `BackColor` de los botones deberían generarse aleatoriamente al inicio del juego, lo cual no está explícitamente en el código proporcionado pero es una extensión lógica.

Programa 96. Juego de laberinto simple

Este programa simula un juego de laberinto donde el usuario controla un "jugador" con las teclas de dirección para alcanzar una "meta".

- **Controles necesarios en el formulario:**
    - `Panel`: **Name:** `pnlLaberinto` (se usaría para dibujar las paredes, aunque el código solo se enfoca en el jugador y la meta).
    - `Button`: **Name:** `btnJugador`, **Text:** "J" (para el jugador).
    - `Button`: **Name:** `btnMeta`, **Text:** "M" (para la meta).
- **Configuración adicional:**
    - La propiedad **`KeyPreview`** del formulario debe establecerse a `True` para que el formulario capture los eventos de teclado antes que los controles.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnJugador_KeyDown(sender As Object, e As KeyEventArgs)
Handles Me.KeyDown
        Select Case e.KeyCode
            Case Keys.Up : btnJugador.Top -= 5
            Case Keys.Down : btnJugador.Top += 5
            Case Keys.Left : btnJugador.Left -= 5
            Case Keys.Right : btnJugador.Left += 5
        End Select
        If btnJugador.Bounds.IntersectsWith(btnMeta.Bounds) Then
            MessageBox.Show("¡Has ganado!")
        End If
    End Sub
End Class
```

- **Explicación:**
    - El evento `btnJugador_KeyDown` (`Me.KeyDown` en el código, pero el programa 106 usa `Form1_KeyDown` que es lo mismo cuando `KeyPreview=True`) se dispara cuando el usuario presiona una tecla mientras el formulario tiene el foco.
    - Un bloque `Select Case` evalúa qué tecla de dirección (`Keys.Up`, `Keys.Down`, `Keys.Left`, `Keys.Right`) fue presionada.
    - Dependiendo de la tecla, las propiedades `Top` o `Left` de `btnJugador` se ajustan en 5 píxeles, moviendo el botón en la dirección correspondiente.
    - Después de cada movimiento, `btnJugador.Bounds.IntersectsWith(btnMeta.Bounds)` verifica si los límites del `btnJugador` se superponen con los límites del `btnMeta`. Si lo hacen, significa que el jugador ha alcanzado la meta y se muestra un mensaje de "¡Has ganado!".

Programa 97. Juego de dados con imágenes

Este programa simula el lanzamiento de dos dados y muestra los resultados con imágenes.

- **Controles necesarios en el formulario:**
  - `PictureBox`: **Name:** `picDado1`, `picDado2`.
  - `Button`: **Name:** `btnTirar`, **Text:** "Tirar Dados".
- **Recursos adicionales:**
  - Se requiere tener imágenes `dado1.png` a `dado6.png` (o `.jpg`) agregadas a los **recursos del proyecto** (en `My Project > Resources`).
- **Código VB.NET:**

```
Public Class Form1
    Dim rnd As New Random()
    Private Sub btnTirar_Click(sender As Object, e As EventArgs) Handles
btnTirar.Click
        Dim d1 As Integer = rnd.Next(1, 7)
        Dim d2 As Integer = rnd.Next(1, 7)
        picDado1.Image = My.Resources.ResourceManager.GetObject("dado" &
d1)
        picDado2.Image = My.Resources.ResourceManager.GetObject("dado" &
d2)
    End Sub
End Class
```

- **Explicación:**
  - Se declara una instancia de `Random` (`rnd`) a nivel de clase para generar números aleatorios.
  - Cuando se hace clic en `btnTirar`, se generan dos números aleatorios `d1` y `d2` entre 1 y 6 (inclusive) utilizando `rnd.Next(1, 7)`.
  - `My.Resources.ResourceManager.GetObject("dado" & d1)` construye el nombre del recurso de imagen (ej. "dado1", "dado2") y carga la imagen correspondiente desde los recursos del proyecto.
  - Estas imágenes se asignan a las propiedades `Image` de `picDado1` y `picDado2`, mostrando los resultados visualmente.

Programa 98. Juego de ahorcado

Este programa implementa una versión simple del juego del ahorcado.

- **Controles necesarios en el formulario:**
    - `Label`: **Name:** `lblPalabra`.
    - `TextBox`: **Name:** `txtLetra`.
    - `Button`: **Name:** `btnProbar`, **Text:** "Probar".
    - `Label`: **Name:** `lblIntentos`.
- **Código VB.NET:**

```
Public Class Form1
    Dim palabra As String = "VBNET"
    Dim adivinada As String = "_____"
    Dim intentos As Integer = 6
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        lblPalabra.Text = adivinada
        lblIntentos.Text = "Intentos: " & intentos
    End Sub
    Private Sub btnProbar_Click(sender As Object, e As EventArgs) Handles
btnProbar.Click
        Dim letra As String = txtLetra.Text.ToUpper()
        Dim nueva As Char() = adivinada.ToCharArray()
        For i As Integer = 0 To palabra.Length - 1
            If palabra(i) = letra Then
                nueva(i) = letra
            End If
        Next
        adivinada = New String(nueva)
        lblPalabra.Text = adivinada
        If Not palabra.Contains(letra) Then
            intentos -= 1
            lblIntentos.Text = "Intentos: " & intentos
        End If
        If adivinada = palabra Then MessageBox.Show("¡Ganaste!")
        If intentos = 0 Then MessageBox.Show("¡Perdiste!")
        txtLetra.Clear()
    End Sub
End Class
```

- **Explicación:**
    - Variables a nivel de clase `palabra` (la palabra oculta), `adivinada` (la palabra con guiones bajos que el usuario va completando) e `intentos` (vidas restantes).
    - En `Form1_Load`, se inicializan `lblPalabra` y `lblIntentos` con los valores iniciales.
    - Cuando se hace clic en `btnProbar`, se obtiene la `letra` ingresada y se convierte a mayúscula.
    - Se crea un array de caracteres `nueva` a partir de `adivinada`. Se itera por `palabra`: si la `letra` coincide, se actualiza la posición correspondiente en `nueva`.
    - `adivinada` se actualiza con los cambios, y `lblPalabra` muestra el progreso.
    - Si la `letra` no está contenida en `palabra`, los `intentos` se decrementan y `lblIntentos` se actualiza.

- Se verifica si el jugador ha ganado (`adivinada = palabra`) o ha perdido (`intentos = 0`) y se muestra un `MessageBox` apropiado. Finalmente, `txtLetra` se limpia.

Programa 99. Juego de tic-tac-toe

Este programa implementa el juego de tres en raya (tic-tac-toe) para dos jugadores, donde se alternan los turnos.

- **Controles necesarios en el formulario:**
  - `9 Buttons`: **Name:** `btn1` a `btn9`, **Text:** `""` (vacío al inicio).
- **Código VB.NET:**

```
Public Class Form1
    Dim turno As Boolean = True ' True = X, False = O
    Private Sub btn_Click(sender As Object, e As EventArgs) Handles
btn1.Click, btn2.Click, btn3.Click, btn4.Click, btn5.Click, btn6.Click,
btn7.Click, btn8.Click, btn9.Click
        Dim btn As Button = CType(sender, Button)
        If btn.Text = "" Then
            btn.Text = If(turno, "X", "O")
            turno = Not turno
            ' Opcional: agregar función para revisar ganador
        End If
    End Sub
End Class
```

- **Explicación:**
  - La variable `turno` (a nivel de clase) controla quién juega: `True` para 'X' y `False` para 'O'.
  - El evento `btn_Click` se maneja para los nueve botones de juego. Se convierte el `sender` al tipo `Button`.
  - Se verifica si el `btn.Text` está vacío, lo que indica que la casilla no ha sido jugada.
  - Si está vacía, se asigna 'X' o 'O' a `btn.Text` según el valor de `turno`, y luego `turno` se invierte (`Not turno`) para el siguiente jugador.
  - **Nota:** El código proporcionado solo maneja los turnos; no incluye la lógica para determinar un ganador o un empate.

Programa 100. Juego de cartas simple

Este programa simula sacar una carta aleatoria de una baraja y mostrarla.

- **Controles necesarios en el formulario:**
  - `PictureBox`: **Name:** `picCarta`.
  - `Button`: **Name:** `btnSacar`, **Text:** "Sacar Carta".
- **Recursos adicionales:**
  - Se requiere agregar imágenes `c1.png` a `c5.png` (o `.jpg`) en los **recursos del proyecto** (`My Project > Resources`).
- **Código VB.NET:**

```
Public Class Form1
    Dim rnd As New Random()
    Dim cartas() As String = {"c1", "c2", "c3", "c4", "c5"}
    Private Sub btnSacar_Click(sender As Object, e As EventArgs) Handles
btnSacar.Click
        Dim index As Integer = rnd.Next(cartas.Length)
        picCarta.Image =
My.Resources.ResourceManager.GetObject(cartas(index))
    End Sub
End Class
```

- **Explicación:**
  - Una instancia de `Random` (`rnd`) se utiliza para generar aleatoriedad, y un array `cartas` contiene los nombres de las imágenes de las cartas.
  - Cuando se hace clic en `btnSacar`, se genera un `index` aleatorio dentro del rango del array `cartas`.
  - `My.Resources.ResourceManager.GetObject(cartas(index))` recupera la imagen correspondiente del recurso del proyecto utilizando el nombre de la carta aleatoria, y la asigna a `picCarta.Image`.

Programa 101. Juego de lotería con números aleatorios

Este programa genera un conjunto de 6 números aleatorios para simular un sorteo de lotería.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnGenerar`, **Text:** "Generar Números".
  - `ListBox`: **Name:** `lstNumeros`.
- **Código VB.NET:**

```
Public Class Form1
    Dim rnd As New Random()
    Private Sub btnGenerar_Click(sender As Object, e As EventArgs) Handles
btnGenerar.Click
        lstNumeros.Items.Clear()
        For i As Integer = 1 To 6
            lstNumeros.Items.Add(rnd.Next(1, 50))
        Next
    End Sub
End Class
```

- **Explicación:**
  - Se declara una instancia de `Random` (`rnd`) a nivel de clase.
  - Al hacer clic en `btnGenerar`, `lstNumeros` se limpia de cualquier número anterior.
  - Un bucle `For` se ejecuta 6 veces. En cada iteración, `rnd.Next(1, 50)` genera un número entero aleatorio entre 1 (inclusive) y 50 (exclusivo, es decir, del 1 al 49), y lo añade a `lstNumeros`.

Programa 102. Juego de ruleta simple

Este programa simula un giro de ruleta, generando un número aleatorio entre 0 y 36.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnGirar`, **Text:** "Girar".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Dim rnd As New Random()
    Private Sub btnGirar_Click(sender As Object, e As EventArgs) Handles
btnGirar.Click
        Dim numero As Integer = rnd.Next(0, 37)
        lblResultado.Text = "Número: " & numero
    End Sub
End Class
```

- **Explicación:**
  - Una instancia de `Random` (`rnd`) se utiliza para generar el número.
  - Cuando se hace clic en `btnGirar`, `rnd.Next(0, 37)` genera un número entero aleatorio entre 0 (inclusive) y 37 (exclusivo), lo que simula los números de una ruleta (0 a 36).
  - El número generado se muestra en `lblResultado.Text`.

Programa 103. Juego de preguntas y respuestas

Este programa presenta una pregunta y verifica si la respuesta ingresada por el usuario es correcta.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblPregunta`.
  - `TextBox`: **Name:** `txtRespuesta`.
  - `Button`: **Name:** `btnVerificar`, **Text:** "Verificar".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Public Class Form1
    Dim pregunta As String = "Capital de Francia?"
    Dim respuestaCorrecta As String = "París"
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        lblPregunta.Text = pregunta
    End Sub
    Private Sub btnVerificar_Click(sender As Object, e As EventArgs)
Handles btnVerificar.Click
        If txtRespuesta.Text.ToLower() = respuestaCorrecta.ToLower() Then
            lblResultado.Text = "Correcto"
        Else
            lblResultado.Text = "Incorrecto"
        End If
    End Sub
End Class
```

- **Explicación:**
  - Las variables `pregunta` y `respuestaCorrecta` almacenan la pregunta y su solución, respectivamente.
  - En `Form1_Load`, `lblPregunta` se inicializa con el texto de la pregunta.
  - Cuando se hace clic en `btnVerificar`, el texto de `txtRespuesta` se compara (sin distinguir mayúsculas/minúsculas con `.ToLower()`) con `respuestaCorrecta`.
  - `lblResultado.Text` se actualiza para indicar si la respuesta fue "Correcto" o "Incorrecto".

Programa 104. Juego de carreras con imágenes

Este programa simula una carrera entre dos "carros" representados por `PictureBox`s, moviéndose aleatoriamente hacia la derecha.

- **Controles necesarios en el formulario:**
  - `PictureBox`: **Name:** `picCarro1`, `picCarro2`.
  - `Button`: **Name:** `btnCorrer`, **Text:** "Correr".
  - `Timer`: **Name:** `Timer1`, **Interval:** `50` (milisegundos).
- **Código VB.NET:**

```
Public Class Form1
    Dim rnd As New Random()
    Private Sub btnCorrer_Click(sender As Object, e As EventArgs) Handles
btnCorrer.Click
        Timer1.Start()
    End Sub
    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
        picCarro1.Left += rnd.Next(1, 10)
        picCarro2.Left += rnd.Next(1, 10)
        If picCarro1.Right >= Me.Width Or picCarro2.Right >= Me.Width Then
            Timer1.Stop()
            MessageBox.Show("Carrera terminada")
        End If
    End Sub
End Class
```

- **Explicación:**
  - Se utiliza una instancia de `Random` (`rnd`) para la aleatoriedad en el movimiento.
  - Al hacer clic en `btnCorrer`, `Timer1` se inicia.
  - Cada 50 milisegundos (`Timer1_Tick`), `picCarro1.Left` y `picCarro2.Left` se incrementan en una cantidad aleatoria entre 1 y 9 píxeles (`rnd.Next(1, 10)`), moviendo los carros hacia la derecha.
  - Se comprueba si alguno de los carros (`picCarro1.Right` o `picCarro2.Right`) ha alcanzado o superado el borde derecho del formulario (`Me.Width`). Si es así, la carrera termina (`Timer1.Stop()`) y se muestra un `MessageBox`.

Programa 105. Juego de disparos con botones

Este es un juego de disparos conceptual muy simple donde el usuario "dispara" haciendo clic en un botón, y un `MessageBox` simula si el disparo fue un "acierto" o no.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnDisparar`, **Text:** "Disparar".
  - `Label`: **Name:** `lblPuntaje`.
- **Código VB.NET:**

```
Public Class Form1
    Dim puntaje As Integer = 0
    Private Sub btnDisparar_Click(sender As Object, e As EventArgs)
Handles btnDisparar.Click
        If MessageBox.Show("Acertaste?", "Disparo",
MessageBoxButtons.YesNo) = DialogResult.Yes Then
            puntaje += 1
        End If
        lblPuntaje.Text = "Puntaje: " & puntaje
    End Sub
End Class
```

- **Explicación:**
  - La variable `puntaje` a nivel de clase mantiene la puntuación del jugador.
  - Cuando se hace clic en `btnDisparar`, se muestra un `MessageBox` de tipo `YesNo` preguntando "¿Acertaste?".
  - Si el usuario selecciona "Sí" (`DialogResult.Yes`), el `puntaje` se incrementa en 1.
  - Finalmente, `lblPuntaje.Text` se actualiza para mostrar el puntaje actual.

Programa 106. Juego de laberinto con teclas de dirección

Este programa es idéntico al Programa 96 y simula un juego de laberinto donde el usuario controla un "jugador" con las teclas de dirección para alcanzar una "meta".

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnJugador`, **Text:** "J".
  - `Panel`: **Name:** `pnlLaberinto` (paredes).
  - `Button`: **Name:** `btnMeta`, **Text:** "M".
- **Configuración adicional:**
  - La propiedad **`KeyPreview`** del formulario debe establecerse a `True` para que el formulario capture los eventos de teclado antes que los controles.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub Form1_KeyDown(sender As Object, e As KeyEventArgs) Handles
Me.KeyDown
        Select Case e.KeyCode
            Case Keys.Up : btnJugador.Top -= 5
            Case Keys.Down : btnJugador.Top += 5
            Case Keys.Left : btnJugador.Left -= 5
            Case Keys.Right : btnJugador.Left += 5
        End Select
        If btnJugador.Bounds.IntersectsWith(btnMeta.Bounds) Then
            MessageBox.Show("¡Ganaste!")
        End If
    End Sub
End Class
```

- **Explicación:**
  - El evento `Form1_KeyDown` se dispara cuando se presiona una tecla mientras el formulario tiene el foco, gracias a que `KeyPreview` está en `True`.
  - Un `Select Case` verifica la tecla presionada (`e.KeyCode`).
  - Las propiedades `Top` o `Left` de `btnJugador` se ajustan en 5 píxeles, moviendo el botón del jugador.
  - Si `btnJugador` se superpone con `btnMeta` (verificado con `btnJugador.Bounds.IntersectsWith(btnMeta.Bounds)`), se muestra un `MessageBox` de victoria.

Programa 107. Agenda con búsqueda por nombre

Este programa implementa una funcionalidad de búsqueda para filtrar contactos en un `ListBox` por nombre.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtBuscar`.
    - `Button`: **Name:** `btnBuscar`, **Text:** "Buscar".
    - `ListBox`: **Name:** `lstContactos`.
- **Código VB.NET:**

```
Public Class Form1
    Dim contactos As New List(Of String) From {"Ana", "Luis", "Pedro"}
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        lstContactos.Items.AddRange(contactos.ToArray())
    End Sub
    Private Sub btnBuscar_Click(sender As Object, e As EventArgs) Handles
btnBuscar.Click
        Dim filtro = txtBuscar.Text.ToLower()
        lstContactos.Items.Clear()
        For Each c In contactos
            If c.ToLower().Contains(filtro) Then lstContactos.Items.Add(c)
        Next
    End Sub
End Class
```

- **Explicación:**
    - La lista `contactos` (a nivel de clase) almacena todos los nombres.
    - En `Form1_Load`, los contactos iniciales se añaden a `lstContactos`.
    - Cuando se hace clic en `btnBuscar`, se obtiene el texto del `txtBuscar` y se convierte a minúsculas (`.ToLower()`) para el `filtro`.
    - `lstContactos` se vacía, y luego se itera sobre la lista `contactos` original. Cada contacto que contiene el `filtro` (sin importar mayúsculas/minúsculas) se añade de nuevo a `lstContactos`.

Programa 108. Agenda con edición de contactos

Este programa permite editar un contacto seleccionado en un `ListBox` utilizando un `TextBox`.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtContacto`.
  - `Button`: **Name:** `btnEditar`, **Text:** "Editar".
  - `ListBox`: **Name:** `lstContactos`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnEditar_Click(sender As Object, e As EventArgs) Handles
btnEditar.Click
        If lstContactos.SelectedIndex >= 0 Then
            lstContactos.Items(lstContactos.SelectedIndex) =
txtContacto.Text
        End If
    End Sub
End Class
```

- **Explicación:**
  - El evento `btnEditar_Click` se activa cuando se presiona el botón "Editar".
  - Se verifica si un elemento está seleccionado en `lstContactos` (`lstContactos.SelectedIndex >= 0`).
  - Si hay un elemento seleccionado, `lstContactos.Items(lstContactos.SelectedIndex) = txtContacto.Text` reemplaza el texto del elemento seleccionado con el nuevo texto ingresado en `txtContacto.Text`.

Programa 109. Agenda con borrado de contactos

Este programa permite borrar un contacto seleccionado de un `ListBox`.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnBorrar`, **Text:** "Borrar".
  - `ListBox`: **Name:** `lstContactos`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnBorrar_Click(sender As Object, e As EventArgs) Handles
btnBorrar.Click
        If lstContactos.SelectedIndex >= 0 Then
            lstContactos.Items.RemoveAt(lstContactos.SelectedIndex)
        End If
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnBorrar`, el código en `btnBorrar_Click` se ejecuta.
  - Se comprueba si hay un elemento seleccionado en `lstContactos`.
  - Si un elemento está seleccionado, `lstContactos.Items.RemoveAt(lstContactos.SelectedIndex)` elimina ese elemento de la lista.

Programa 110. Agenda con exportación a archivo de texto

Este programa permite exportar la lista de contactos de un `ListBox` a un archivo de texto.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnExportar`, **Text:** "Exportar".
  - `ListBox`: **Name:** `lstContactos`.
- **Código VB.NET:**

```
Imports System.IO
Public Class Form1
    Private Sub btnExportar_Click(sender As Object, e As EventArgs)
Handles btnExportar.Click
        Using sw As New StreamWriter("agenda.txt")
            For Each c As String In lstContactos.Items
                sw.WriteLine(c)
            Next
        End Using
        MessageBox.Show("Exportado correctamente")
    End Sub
End Class
```

- **Explicación:**
  - Se importa el espacio de nombres `System.IO` para trabajar con archivos.
  - Cuando se hace clic en `btnExportar`, se crea un `StreamWriter` para escribir en el archivo "agenda.txt".
  - Un bucle `For Each` itera sobre cada elemento en `lstContactos.Items`, y cada elemento se escribe como una nueva línea en el archivo usando `sw.WriteLine(c)`.
  - La declaración `Using` asegura que el `StreamWriter` se cierre y los recursos se liberen automáticamente. Se muestra un `MessageBox` de confirmación.

Programa 111. Agenda con importación desde archivo de texto

Este programa permite importar contactos desde un archivo de texto y cargarlos en un `ListBox`.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnImportar`, **Text:** "Importar".
  - `ListBox`: **Name:** `lstContactos`.
  - `OpenFileDialog`: **Name:** `OpenFileDialog1`.
- **Código VB.NET:**

```
Imports System.IO
Public Class Form1
    Private Sub btnImportar_Click(sender As Object, e As EventArgs)
Handles btnImportar.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            lstContactos.Items.Clear()
            Dim lineas() As String =
File.ReadAllLines(OpenFileDialog1.FileName)
            lstContactos.Items.AddRange(lineas)
        End If
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.IO`.
  - Al hacer clic en `btnImportar`, `OpenFileDialog1.ShowDialog()` abre el cuadro de diálogo para que el usuario seleccione un archivo.
  - Si se selecciona un archivo (`DialogResult.OK`), `lstContactos` se vacía, y `File.ReadAllLines(OpenFileDialog1.FileName)` lee todas las líneas del archivo seleccionado en un array de cadenas.
  - `lstContactos.Items.AddRange(lineas)` añade todas estas líneas como elementos al `ListBox`, actualizando la lista de contactos visible.

Programa 112. Agenda con exportación a Excel

Este programa exporta la lista de contactos a una hoja de cálculo de Microsoft Excel.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnExportarExcel`, **Text:** "Exportar a Excel".
  - `ListBox`: **Name:** `lstContactos`.
- **Requisito adicional:**
  - Requiere agregar una **referencia a la biblioteca de objetos de Microsoft Excel** en el proyecto.
- **Código VB.NET:**

```
Imports Microsoft.Office.Interop.Excel
Public Class Form1
    Private Sub btnExportarExcel_Click(sender As Object, e As EventArgs)
Handles btnExportarExcel.Click
        Dim app As New Application
        Dim wb = app.Workbooks.Add()
        Dim ws = CType(wb.Sheets(1), Worksheet)
        For i As Integer = 0 To lstContactos.Items.Count - 1
            ws.Cells(i + 1, 1) = lstContactos.Items(i)
        Next
        app.Visible = True
    End Sub
End Class
```

- **Explicación:**
  - Se importa el espacio de nombres `Microsoft.Office.Interop.Excel`.
  - Al hacer clic en `btnExportarExcel`, se crea una nueva instancia de la aplicación Excel (`app`).
  - Se añade un nuevo libro de trabajo (`wb`) y se obtiene la primera hoja de trabajo (`ws`).
  - Un bucle itera sobre los elementos de `lstContactos`, y cada elemento se escribe en la primera columna de Excel (`ws.Cells(i + 1, 1)`).
  - Finalmente, `app.Visible = True` hace visible la aplicación Excel con los datos exportados.

Programa 113. Agenda con importación desde Excel

Este programa importa contactos desde una hoja de cálculo de Microsoft Excel a un `ListBox`.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnImportarExcel`, **Text:** "Importar desde Excel".
  - `ListBox`: **Name:** `lstContactos`.
  - `OpenFileDialog`: **Name:** `OpenFileDialog1`.
- **Requisito adicional:**
  - Requiere agregar una **referencia a la biblioteca de objetos de Microsoft Excel** en el proyecto.
- **Código VB.NET:**

```
Imports Microsoft.Office.Interop.Excel
Public Class Form1
    Private Sub btnImportarExcel_Click(sender As Object, e As EventArgs)
Handles btnImportarExcel.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            Dim app As New Application
            Dim wb = app.Workbooks.Open(OpenFileDialog1.FileName)
            Dim ws = CType(wb.Sheets(1), Worksheet)
            lstContactos.Items.Clear()
            Dim fila As Integer = 1
            While ws.Cells(fila, 1).Value IsNot Nothing
                lstContactos.Items.Add(ws.Cells(fila, 1).Value)
                fila += 1
            End While
            wb.Close(False)
            app.Quit()
        End If
    End Sub
End Class
```

- **Explicación:**
  - Se importa `Microsoft.Office.Interop.Excel`.
  - Al hacer clic en `btnImportarExcel`, se abre `OpenFileDialog1` para seleccionar el archivo Excel.
  - Si se selecciona un archivo, se crea una instancia de la aplicación Excel (`app`) y se abre el libro de trabajo (`wb`).
  - Se obtiene la primera hoja de trabajo (`ws`), y `lstContactos` se vacía.
  - Un bucle `While` lee los valores de la primera columna de Excel fila por fila (`ws.Cells(fila, 1).Value`) mientras no estén vacíos, añadiéndolos a `lstContactos`.
  - Finalmente, el libro de trabajo se cierra (`wb.Close(False)`) y la aplicación Excel se cierra (`app.Quit()`).

Programa 114. Agenda con base de datos SQLite

Este programa utiliza una base de datos SQLite para almacenar y gestionar contactos.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnCargar`, **Text:** "Cargar".
    - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar".
    - `TextBox`: **Name:** `txtContacto`.
    - `ListBox`: **Name:** `lstContactos`.
- **Requisito adicional:**
    - Requiere agregar una referencia a la librería **System.Data.SQLite** (generalmente a través de NuGet).
- **Código VB.NET:**

```
Imports System.Data.SQLite
Public Class Form1
    Dim conn As New SQLiteConnection("Data Source=agenda.db;Version=3;")
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        conn.Open()
        Dim cmd As New SQLiteCommand("CREATE TABLE IF NOT EXISTS contactos
(nombre TEXT)", conn)
        cmd.ExecuteNonQuery()
        CargarContactos()
    End Sub
    Private Sub CargarContactos()
        lstContactos.Items.Clear()
        Dim cmd As New SQLiteCommand("SELECT nombre FROM contactos", conn)
        Dim dr = cmd.ExecuteReader()
        While dr.Read()
            lstContactos.Items.Add(dr("nombre"))
        End While
        ' dr.Close() (implied or should be added for robustness)
    End Sub
    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
        Dim cmd As New SQLiteCommand("INSERT INTO contactos(nombre)
VALUES(@nombre)", conn)
        cmd.Parameters.AddWithValue("@nombre", txtContacto.Text)
        cmd.ExecuteNonQuery()
        CargarContactos()
    End Sub
End Class
```

- **Explicación:**
    - Se importa `System.Data.SQLite`.
    - `conn` es el objeto de conexión a la base de datos SQLite (`agenda.db`).
    - En `Form1_Load`, la conexión se abre. Se crea una tabla `contactos` si no existe. Luego, se llama a `CargarContactos` para poblar `lstContactos` al inicio.
    - `CargarContactos` limpia `lstContactos`, ejecuta una consulta `SELECT` para obtener todos los nombres de la tabla `contactos` y los añade a `lstContactos`.
    - En `btnAgregar_Click`, se crea un comando `INSERT` para añadir un nuevo contacto. Se utiliza un parámetro `@nombre` para evitar inyecciones SQL. Se ejecuta la consulta y se llama a `CargarContactos` para actualizar la lista.

Programa 115. Agenda con base de datos SQL Server

Este programa utiliza una base de datos SQL Server para almacenar y gestionar contactos.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnCargar`, **Text:** "Cargar".
    - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar".
    - `TextBox`: **Name:** `txtContacto`.
    - `ListBox`: **Name:** `lstContactos`.
- **Requisito adicional:**
    - Requiere tener un servidor SQL Server instalado y una base de datos llamada "Agenda" con una tabla "contactos" que contenga una columna "nombre".
- **Código VB.NET:**

```
Imports System.Data.SqlClient
Public Class Form1
    Dim conn As New
SqlConnection("Server=localhost;Database=Agenda;Trusted_Connection=True;")
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        conn.Open()
        CargarContactos()
    End Sub
    Private Sub CargarContactos()
        lstContactos.Items.Clear()
        Dim cmd As New SqlCommand("SELECT nombre FROM contactos", conn)
        Dim dr = cmd.ExecuteReader()
        While dr.Read()
            lstContactos.Items.Add(dr("nombre"))
        End While
        dr.Close()
    End Sub
    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
        Dim cmd As New SqlCommand("INSERT INTO contactos(nombre)
VALUES(@nombre)", conn)
        cmd.Parameters.AddWithValue("@nombre", txtContacto.Text)
        cmd.ExecuteNonQuery()
        CargarContactos()
    End Sub
End Class
```

- **Explicación:**
    - Se importa `System.Data.SqlClient`.
    - `conn` es el objeto de conexión a SQL Server, apuntando a una base de datos "Agenda" en `localhost` con autenticación de confianza.
    - En `Form1_Load`, la conexión se abre y se llama a `CargarContactos`.
    - `CargarContactos` limpia `lstContactos`, ejecuta un `SELECT` para obtener nombres de `contactos`, y los añade a `lstContactos`. Es importante cerrar el `DataReader` (`dr.Close()`).
    - `btnAgregar_Click` crea un comando `INSERT` parametrizado, añade el nombre, ejecuta la consulta y recarga la lista.

Programa 116. Agenda con base de datos MySQL

Este programa utiliza una base de datos MySQL para almacenar y gestionar contactos.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnCargar`, **Text:** "Cargar".
    - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar".
    - `TextBox`: **Name:** `txtContacto`.
    - `ListBox`: **Name:** `lstContactos`.
- **Requisito adicional:**
    - Requiere agregar una referencia al conector **MySql.Data** (generalmente a través de NuGet) y tener un servidor MySQL con una base de datos "agenda" y una tabla "contactos".
- **Código VB.NET:**

```
Imports MySql.Data.MySqlClient
Public Class Form1
    Dim conn As New
MySqlConnection("server=localhost;user=root;database=agenda;password=;")
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        conn.Open()
        CargarContactos()
    End Sub
    Private Sub CargarContactos()
        lstContactos.Items.Clear()
        Dim cmd As New MySqlCommand("SELECT nombre FROM contactos", conn)
        Dim dr = cmd.ExecuteReader()
        While dr.Read()
            lstContactos.Items.Add(dr("nombre"))
        End While
        dr.Close()
    End Sub
    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
        Dim cmd As New MySqlCommand("INSERT INTO contactos(nombre)
VALUES(@nombre)", conn)
        cmd.Parameters.AddWithValue("@nombre", txtContacto.Text)
        cmd.ExecuteNonQuery()
        CargarContactos()
    End Sub
End Class
```

- **Explicación:**
    - Se importa `MySql.Data.MySqlClient`.
    - `conn` es el objeto de conexión a MySQL, apuntando a una base de datos "agenda" en `localhost` con usuario `root` y una contraseña vacía (ajustar según la configuración).
    - En `Form1_Load`, la conexión se abre y se llama a `CargarContactos`.
    - `CargarContactos` funciona de manera similar a los otros ejemplos de base de datos, limpiando el `ListBox`, ejecutando un `SELECT` y añadiendo los resultados. Es crucial cerrar el `DataReader`.

- `btnAgregar_Click` inserta un nuevo contacto utilizando un comando `INSERT` parametrizado y actualiza la lista.

Programa 117. Agenda con base de datos PostgreSQL

Este programa utiliza una base de datos PostgreSQL para almacenar y gestionar contactos.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnCargar`, **Text:** "Cargar".
    - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar".
    - `TextBox`: **Name:** `txtContacto`.
    - `ListBox`: **Name:** `lstContactos`.
- **Requisito adicional:**
    - Requiere agregar una referencia al conector **Npgsql** (generalmente a través de NuGet) y tener un servidor PostgreSQL con una base de datos "agenda" y una tabla "contactos".
- **Código VB.NET:**

```
Imports Npgsql
Public Class Form1
    Dim conn As New
NpgsqlConnection("Host=localhost;Username=postgres;Password=;Database=agen
da")
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        conn.Open()
        CargarContactos()
    End Sub
    Private Sub CargarContactos()
        lstContactos.Items.Clear()
        Dim cmd As New NpgsqlCommand("SELECT nombre FROM contactos", conn)
        Dim dr = cmd.ExecuteReader()
        While dr.Read()
            lstContactos.Items.Add(dr("nombre"))
        End While
        dr.Close()
    End Sub
    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
        Dim cmd As New NpgsqlCommand("INSERT INTO contactos(nombre)
VALUES(@nombre)", conn)
        cmd.Parameters.AddWithValue("@nombre", txtContacto.Text)
        cmd.ExecuteNonQuery()
        CargarContactos()
    End Sub
End Class
```

- **Explicación:**
    - Se importa `Npgsql`.
    - `conn` es el objeto de conexión a PostgreSQL, con la cadena de conexión apuntando a un host local, usuario `postgres` y una contraseña vacía (ajustar según la configuración) y la base de datos "agenda".
    - En `Form1_Load`, la conexión se abre y se llama a `CargarContactos`.
    - `CargarContactos` limpia el `ListBox`, ejecuta un `SELECT` para recuperar los nombres de la tabla `contactos` y los añade a `lstContactos`. El `DataReader` se cierra.

- `btnAgregar_Click` inserta un nuevo contacto usando un comando `INSERT` parametrizado y recarga los contactos.

Programa 118. Aplicación de notas con recordatorio

Este programa permite al usuario guardar notas y establecer un recordatorio para cada una, mostrando un `MessageBox` cuando la fecha/hora del recordatorio se cumple.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtNota`, **Multiline:** `True`.
    - `Button`: **Name:** `btnGuardar`, **Text:** "Guardar Nota".
    - `DateTimePicker`: **Name:** `dtpRecordatorio`.
    - `Timer`: **Name:** `Timer1`, **Interval:** `1000` (milisegundos).
- **Código VB.NET:**

```
Public Class Form1
    Dim notas As New List(Of Tuple(Of String, DateTime))
    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click
        notas.Add(Tuple.Create(txtNota.Text, dtpRecordatorio.Value))
    End Sub
    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
        For Each nota In notas
            If DateTime.Now >= nota.Item2 Then
                MessageBox.Show("Recordatorio: " & nota.Item1)
            End If
        Next
    End Sub
End Class
```

- **Explicación:**
    - `notas` es una lista de `Tuple(Of String, DateTime)`, donde cada tupla almacena el texto de la nota (`Item1`) y la fecha/hora del recordatorio (`Item2`).
    - En `btnGuardar_Click`, el texto de `txtNota` y la fecha/hora seleccionada en `dtpRecordatorio` se añaden como una nueva tupla a la lista `notas`.
    - El `Timer1` debe estar habilitado al inicio (o habilitarse en `Form_Load`). En cada `Timer1_Tick` (cada segundo), se itera sobre la lista `notas`.
    - Si la hora actual (`DateTime.Now`) es igual o posterior a la hora del recordatorio (`nota.Item2`), se muestra un `MessageBox` con el texto de la nota.

Programa 119. Aplicación de calendario simple

Este programa utiliza el control `MonthCalendar` para mostrar un calendario y permite al usuario ver la fecha seleccionada.

- **Controles necesarios en el formulario:**
  - `MonthCalendar`: **Name:** `MonthCalendar1`.
  - `Label`: **Name:** `lblFecha`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub MonthCalendar1_DateChanged(sender As Object, e As
DateRangeEventArgs) Handles MonthCalendar1.DateChanged
        lblFecha.Text = "Fecha seleccionada: " &
e.Start.ToShortDateString()
    End Sub
End Class
```

- **Explicación:**
  - El evento `MonthCalendar1_DateChanged` se dispara cada vez que el usuario selecciona una nueva fecha en el `MonthCalendar`.
  - `e.Start` devuelve la fecha de inicio del rango seleccionado (que para una selección simple es la fecha seleccionada).
  - `e.Start.ToShortDateString()` convierte esta fecha a un formato de cadena corto (ej. "DD/MM/AAAA").
  - `lblFecha.Text` se actualiza para mostrar la fecha seleccionada.

Programa 120. Agenda con calendario integrado

Este programa combina un calendario con una funcionalidad de notas, permitiendo al usuario agregar notas a fechas específicas y visualizarlas al seleccionar una fecha.

- **Controles necesarios en el formulario:**
  - `MonthCalendar`: **Name:** `mcAgenda`.
  - `ListBox`: **Name:** `lstNotas`.
  - `TextBox`: **Name:** `txtNota`.
  - `Button`: **Name:** `btnAgregarNota`, **Text:** "Agregar Nota".
- **Código VB.NET:**

```
Public Class Form1
    Dim notas As New Dictionary(Of Date, List(Of String))
    Private Sub btnAgregarNota_Click(sender As Object, e As EventArgs)
Handles btnAgregarNota.Click
        Dim fecha As Date = mcAgenda.SelectionStart
        If Not notas.ContainsKey(fecha) Then notas(fecha) = New List(Of
String)
        notas(fecha).Add(txtNota.Text)
        ActualizarLista(fecha)
    End Sub
    Private Sub mcAgenda_DateChanged(sender As Object, e As
DateRangeEventArgs) Handles mcAgenda.DateChanged
        ActualizarLista(e.Start)
    End Sub
    Private Sub ActualizarLista(fecha As Date)
        lstNotas.Items.Clear()
        If notas.ContainsKey(fecha) Then
            For Each n In notas(fecha)
                lstNotas.Items.Add(n)
            Next
        End If
    End Sub
End Class
```

- **Explicación:**
  - `notas` es un `Dictionary` que mapea una `Date` (fecha) a una `List(Of String)` de notas, permitiendo almacenar múltiples notas por fecha.
  - En `btnAgregarNota_Click`, se obtiene la fecha seleccionada del `mcAgenda`. Si no hay entradas para esa fecha en el diccionario `notas`, se crea una nueva lista de cadenas. Luego, el texto de `txtNota` se añade a la lista de notas de esa fecha. Finalmente, `ActualizarLista` se llama para mostrar las notas de la fecha actual.
  - El evento `mcAgenda_DateChanged` se dispara cuando el usuario cambia la fecha en el calendario. Simplemente llama a `ActualizarLista` con la nueva fecha seleccionada (`e.Start`).
  - El método `ActualizarLista` limpia `lstNotas` y, si hay notas para la `fecha` especificada en el diccionario `notas`, las añade a `lstNotas` para que sean visibles.

Programa 121. Agenda con alarma

Este programa permite al usuario agregar notas con una fecha y hora de recordatorio. Cuando el momento de la alarma llega, se muestra un mensaje.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtNota`.
  - `DateTimePicker`: **Name:** `dtpAlarma`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Alarma".
  - `Timer`: **Name:** `Timer1`, **Interval:** `1000` (milisegundos).
  - `ListBox`: **Name:** `lstAlarmas`.
- **Código VB.NET:**

```
Public Class Form1
    Dim alarmas As New List(Of Tuple(Of String, DateTime)) '
    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
        alarmas.Add(Tuple.Create(txtNota.Text, dtpAlarma.Value)) '
        lstAlarmas.Items.Add(txtNota.Text & " - " &
dtpAlarma.Value.ToShortTimeString()) '
    End Sub
    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
        For Each alarma In alarmas '
            If DateTime.Now >= alarma.Item2 Then '
                MessageBox.Show("Alarma: " & alarma.Item1) '
                alarmas.Remove(alarma) '
                Exit For '
            End If
        Next
    End Sub
End Class
```

- **Explicación:**
  - Se declara una lista `alarmas` a nivel de clase para almacenar tuplas, donde cada tupla contiene el texto de la nota (`String`) y la fecha/hora de la alarma (`DateTime`).
  - Cuando el usuario hace clic en `btnAgregar`, la nota y la hora seleccionada en `dtpAlarma` se añaden a la lista `alarmas`, y también se muestra en `lstAlarmas`.
  - El `Timer1`, configurado para dispararse cada segundo, verifica en su evento `Timer1_Tick` si la fecha y hora actuales (`DateTime.Now`) han alcanzado o superado la fecha y hora de alguna alarma guardada (`alarma.Item2`).
  - Si se cumple la condición, se muestra un `MessageBox` con el mensaje de la alarma, y luego esa alarma se elimina de la lista para evitar que se repita.

Programa 122. Agenda con notificaciones

Este programa gestiona una lista de notas y muestra una notificación periódica sobre la cantidad de notas pendientes.

- **Controles necesarios en el formulario:**
  - `ListBox`: **Name:** `lstNotas`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Nota".
  - `TextBox`: **Name:** `txtNota`.
  - `Timer`: **Name:** `Timer1`, **Interval:** `60000` (1 minuto).
- **Código VB.NET:**

```
Public Class Form1
    Dim notas As New List(Of String) '
    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
        notas.Add(txtNota.Text) '
        lstNotas.Items.Add(txtNota.Text) '
    End Sub
    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
        If notas.Count > 0 Then '
            MessageBox.Show("Tienes " & notas.Count & " nota(s)
pendiente(s)") '
        End If
    End Sub
End Class
```

- **Explicación:**
  - Se utiliza una lista `notas` para almacenar las notas de tipo `String`.
  - Cuando se hace clic en `btnAgregar`, el texto del `txtNota` se añade a la lista `notas` y también al `lstNotas` visible en el formulario.
  - El `Timer1`, con un intervalo de 60 segundos (1 minuto), comprueba en su evento `Timer1_Tick` si la lista `notas` contiene elementos (`notas.Count > 0`). Si es así, se muestra un `MessageBox` informando al usuario sobre la cantidad de notas pendientes.

Programa 123. Agenda con múltiples formularios

Este programa demuestra cómo abrir una nueva ventana (`Form2`) desde un formulario principal (`Form1`).

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnAbrirForm2`, **Text:** "Abrir Formulario 2".
- **Configuración adicional:**
  - Se debe crear un segundo formulario en el proyecto llamado `Form2`. Puede estar vacío o contener un `Label`.
- **Código VB.NET:**

```
' Form1
Public Class Form1
    Private Sub btnAbrirForm2_Click(sender As Object, e As EventArgs)
Handles btnAbrirForm2.Click
        Dim f2 As New Form2() '
        f2.Show() '
    End Sub
End Class
' Form2 (en su propio archivo)
Public Class Form2
    ' Puede estar vacío
End Class
```

- **Explicación:**
  - Al hacer clic en `btnAbrirForm2`, se ejecuta su evento `btnAbrirForm2_Click`.
  - `Dim f2 As New Form2()` crea una nueva instancia del `Form2`.
  - `f2.Show()` hace visible esta nueva instancia del formulario en la pantalla, abriendo una ventana independiente.

Programa 124. Agenda con login de usuario

Este programa simula un proceso de inicio de sesión con un nombre de usuario y contraseña predefinidos.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtUsuario`.
  - `TextBox`: **Name:** `txtPassword`, **PasswordChar:** *.
  - `Button`: **Name:** `btnLogin`, **Text:** "Login".
  - `Label`: **Name:** `lblMensaje`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
        If txtUsuario.Text = "admin" And txtPassword.Text = "1234" Then '
            lblMensaje.Text = "Acceso concedido" '
        Else
            lblMensaje.Text = "Usuario o contraseña incorrecta" '
        End If
    End Sub
End Class
```

- **Explicación:**
  - Cuando se hace clic en `btnLogin`, el código compara el texto de `txtUsuario` con "admin" y el texto de `txtPassword` con "1234".
  - Si ambas condiciones son verdaderas, `lblMensaje.Text` muestra "Acceso concedido". De lo contrario, muestra "Usuario o contraseña incorrecta". El `PasswordChar: "*"` en `txtPassword` oculta el texto mientras se escribe.

Programa 125. Agenda con niveles de usuario

Este programa demuestra cómo controlar el acceso a la aplicación o ciertas funcionalidades según el nivel de usuario seleccionado en un `ComboBox`.

- **Controles necesarios en el formulario:**
    - `ComboBox`: **Name:** `cmbNivel`.
    - `TextBox`: **Name:** `txtUsuario`.
    - `Button`: **Name:** `btnLogin`, **Text:** "Login".
    - `Label`: **Name:** `lblMensaje`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        cmbNivel.Items.AddRange(New String() {"Usuario", "Administrador"})
'
        cmbNivel.SelectedIndex = 0 '
    End Sub
    Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
        If cmbNivel.SelectedItem = "Administrador" Then '
            lblMensaje.Text = "Acceso de administrador" '
        Else
            lblMensaje.Text = "Acceso de usuario normal" '
        End If
    End Sub
End Class
```

- **Explicación:**
    - En el evento `Form1_Load`, el `ComboBox` `cmbNivel` se carga con las opciones "Usuario" y "Administrador", y "Usuario" se selecciona por defecto.
    - Al hacer clic en `btnLogin`, el programa verifica el `SelectedItem` del `cmbNivel`. Si es "Administrador", `lblMensaje` muestra "Acceso de administrador"; de lo contrario, muestra "Acceso de usuario normal".

Programa 126. Agenda con encriptación de datos

Este programa demuestra una forma básica de encriptar texto utilizando la codificación Base64.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtDato`.
  - `Button`: **Name:** `btnEncriptar`, **Text:** "Encriptar".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Imports System.Text '
Public Class Form1
    Private Sub btnEncriptar_Click(sender As Object, e As EventArgs)
Handles btnEncriptar.Click
        Dim bytes() As Byte = Encoding.UTF8.GetBytes(txtDato.Text) '
        Dim encriptado As String = Convert.ToBase64String(bytes) '
        lblResultado.Text = encriptado '
    End Sub
End Class
```

- **Explicación:**
  - Se importa el espacio de nombres `System.Text` para el manejo de codificaciones.
  - Cuando se hace clic en `btnEncriptar`, el texto de `txtDato` se convierte primero en una matriz de bytes utilizando la codificación UTF-8 (`Encoding.UTF8.GetBytes`).
  - Luego, `Convert.ToBase64String()` convierte esa matriz de bytes en una cadena Base64, que es una representación encriptada simple del texto original. El resultado se muestra en `lblResultado`.

Programa 127. Agenda con desencriptación de datos

Este programa complementa el anterior, mostrando cómo desencriptar una cadena Base64 para recuperar el texto original.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtEncriptado`.
  - `Button`: **Name:** `btnDesencriptar`, **Text:** "Desencriptar".
  - `Label`: **Name:** `lblResultado`.
- **Código VB.NET:**

```
Imports System.Text '
Public Class Form1
    Private Sub btnDesencriptar_Click(sender As Object, e As EventArgs)
Handles btnDesencriptar.Click
        Dim bytes() As Byte = Convert.FromBase64String(txtEncriptado.Text)
'
        Dim original As String = Encoding.UTF8.GetString(bytes) '
        lblResultado.Text = original '
    End Sub
End Class
```

- **Explicación:**
  - También requiere la importación de `System.Text`.
  - Al hacer clic en `btnDesencriptar`, la cadena Base64 de `txtEncriptado` se convierte de nuevo a una matriz de bytes utilizando `Convert.FromBase64String()`.
  - Finalmente, `Encoding.UTF8.GetString()` convierte esa matriz de bytes de nuevo a la cadena de texto original, y el resultado se muestra en `lblResultado`.

Programa 128. Calculadora científica simple

Este programa implementa una calculadora básica con operaciones de suma, resta, multiplicación y división.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtPantalla`.
  - `Buttons`: Para números del 0 al 9 (ej. `btn0` a `btn9`) y operaciones (+, -, *, /) (ej. `btnSumar`, `btnRestar`, `btnMultiplicar`, `btnDividir`).
  - `Button`: **Name:** `btnIgual`, **Text:** =.
- **Código VB.NET:**

```
Public Class Form1
    Dim operacion As String = "" '
    Dim valor1 As Double '
    Private Sub btnNumero_Click(sender As Object, e As EventArgs) Handles
btn0.Click, btn1.Click, btn2.Click, btn3.Click, btn4.Click, btn5.Click,
btn6.Click, btn7.Click, btn8.Click, btn9.Click
        Dim btn As Button = CType(sender, Button) '
        txtPantalla.Text &= btn.Text '
    End Sub
    Private Sub btnOperacion_Click(sender As Object, e As EventArgs)
Handles btnSumar.Click, btnRestar.Click, btnMultiplicar.Click,
btnDividir.Click
        valor1 = Val(txtPantalla.Text) '
        operacion = CType(sender, Button).Text '
        txtPantalla.Clear() '
    End Sub
    Private Sub btnIgual_Click(sender As Object, e As EventArgs) Handles
btnIgual.Click
        Dim valor2 As Double = Val(txtPantalla.Text) '
        Select Case operacion '
            Case "+" : txtPantalla.Text = valor1 + valor2 '
            Case "-" : txtPantalla.Text = valor1 - valor2 '
            Case "*" : txtPantalla.Text = valor1 * valor2 '
            Case "/" : txtPantalla.Text = valor1 / valor2 '
        End Select
    End Sub
End Class
```

- **Explicación:**
  - Las variables `operacion` y `valor1` se declaran a nivel de clase para mantener el estado de la calculadora.
  - Los eventos `btnNumero_Click` (compartido por todos los botones numéricos) añaden el dígito presionado a `txtPantalla.Text`.
  - Los eventos `btnOperacion_Click` (compartido por los botones de operación) guardan el número actual de `txtPantalla` en `valor1`, el símbolo de la operación en `operacion`, y limpian `txtPantalla` para la entrada del segundo número.
  - Cuando se hace clic en `btnIgual`, el valor de `txtPantalla` se guarda como `valor2`. Luego, un `Select Case` realiza la operación almacenada (`operacion`) entre `valor1` y `valor2`, mostrando el resultado en `txtPantalla.Text`.

Programa 129. Calculadora con funciones trigonométricas

Este programa extiende la calculadora con funciones trigonométricas como seno, coseno y tangente.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtPantalla`.
  - `Button`: **Name:** `btnSin`, **Text:** "Sin".
  - `Button`: **Name:** `btnCos`, **Text:** "Cos".
  - `Button`: **Name:** `btnTan`, **Text:** "Tan".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnSin_Click(sender As Object, e As EventArgs) Handles
btnSin.Click
        txtPantalla.Text = Math.Sin(Val(txtPantalla.Text)) '
    End Sub
    Private Sub btnCos_Click(sender As Object, e As EventArgs) Handles
btnCos.Click
        txtPantalla.Text = Math.Cos(Val(txtPantalla.Text)) '
    End Sub
    Private Sub btnTan_Click(sender As Object, e As EventArgs) Handles
btnTan.Click
        txtPantalla.Text = Math.Tan(Val(txtPantalla.Text)) '
    End Sub
End Class
```

- **Explicación:**
  - Cada botón trigonométrico tiene su propio evento `_Click`.
  - Cuando se presiona, el valor numérico en `txtPantalla` se convierte a `Double` con `Val()` y se le aplica la función `Math.Sin()`, `Math.Cos()`, o `Math.Tan()` respectivamente.
  - El resultado de la función trigonométrica reemplaza el texto en `txtPantalla`. (Cabe destacar que `Math.Sin`, `Math.Cos`, `Math.Tan` esperan argumentos en radianes, lo que puede requerir conversión de grados si el usuario ingresa grados).

Programa 130. Calculadora con funciones logarítmicas

Este programa añade funciones logarítmicas a la calculadora: logaritmo base 10 (`Log`) y logaritmo natural (`Ln`).

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtPantalla`.
  - `Button`: **Name:** `btnLog`, **Text:** "Log".
  - `Button`: **Name:** `btnLn`, **Text:** "Ln".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnLog_Click(sender As Object, e As EventArgs) Handles
btnLog.Click
        txtPantalla.Text = Math.Log10(Val(txtPantalla.Text)) '
    End Sub
    Private Sub btnLn_Click(sender As Object, e As EventArgs) Handles
btnLn.Click
        txtPantalla.Text = Math.Log(Val(txtPantalla.Text)) '
    End Sub
End Class
```

- **Explicación:**
  - Cada botón logarítmico tiene un evento `_Click` asociado.
  - Al hacer clic en `btnLog`, `Math.Log10()` calcula el logaritmo en base 10 del número en `txtPantalla` y actualiza la pantalla con el resultado.
  - Al hacer clic en `btnLn`, `Math.Log()` calcula el logaritmo natural (base e) del número en `txtPantalla` y muestra el resultado.

Programa 131. Calculadora con potencias y raíces

Este programa extiende la calculadora para incluir operaciones de potencia (^) y raíz cuadrada (√).

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtPantalla`.
  - `Button`: **Name:** `btnPotencia`, **Text:** "^".
  - `Button`: **Name:** `btnRaiz`, **Text:** "√".
  - `Button`: **Name:** `btnIgual`, **Text:** =.
- **Código VB.NET:**

```
Public Class Form1
    Dim valor1 As Double '
    Dim operacion As String '
    Private Sub btnPotencia_Click(sender As Object, e As EventArgs)
Handles btnPotencia.Click
        valor1 = Val(txtPantalla.Text) '
        operacion = "^" '
        txtPantalla.Clear() '
    End Sub
    Private Sub btnRaiz_Click(sender As Object, e As EventArgs) Handles
btnRaiz.Click
        valor1 = Val(txtPantalla.Text) '
        txtPantalla.Text = Math.Sqrt(valor1) '
    End Sub
    Private Sub btnIgual_Click(sender As Object, e As EventArgs) Handles
btnIgual.Click
        Dim valor2 As Double = Val(txtPantalla.Text) '
        If operacion = "^" Then '
            txtPantalla.Text = Math.Pow(valor1, valor2) '
        End If
        ' (Opcional: agregar otras operaciones aquí si se comparte el
btnIgual)
    End Sub
End Class
```

- **Explicación:**
  - Las variables `valor1` y `operacion` se utilizan para almacenar el primer operando y la operación pendiente.
  - Cuando se hace clic en `btnPotencia`, el valor actual de `txtPantalla` se guarda en `valor1`, `operacion` se establece a "^", y `txtPantalla` se borra para el segundo operando.
  - Al hacer clic en `btnRaiz`, se calcula la raíz cuadrada del valor actual en `txtPantalla` utilizando `Math.Sqrt()` y el resultado se muestra directamente en `txtPantalla`.
  - Cuando se presiona `btnIgual`, si la `operacion` pendiente es "^", `Math.Pow(valor1, valor2)` calcula la potencia, mostrando el resultado en `txtPantalla`.

Programa 132. Calculadora con historial de operaciones

Este programa implementa una calculadora que guarda un historial de las operaciones realizadas en un `ListBox`.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtPantalla`.
  - `ListBox`: **Name:** `lstHistorial`.
  - Botones numéricos y de operaciones (+, -, *, /).
  - `Button`: **Name:** `btnIgual`, **Text:** =.
- **Código VB.NET:**

```
Public Class Form1
    Dim valor1 As Double '
    Dim operacion As String '
    Private Sub btnOperacion_Click(sender As Object, e As EventArgs)
Handles btnSumar.Click, btnRestar.Click, btnMultiplicar.Click,
btnDividir.Click
        valor1 = Val(txtPantalla.Text) '
        operacion = CType(sender, Button).Text '
        txtPantalla.Clear() '
    End Sub
    Private Sub btnIgual_Click(sender As Object, e As EventArgs) Handles
btnIgual.Click
        Dim valor2 As Double = Val(txtPantalla.Text) '
        Dim resultado As Double = 0 '
        Select Case operacion '
            Case "+" : resultado = valor1 + valor2 '
            Case "-" : resultado = valor1 - valor2 '
            Case "*" : resultado = valor1 * valor2 '
            Case "/" : resultado = valor1 / valor2 '
        End Select
        txtPantalla.Text = resultado '
        lstHistorial.Items.Add(valor1 & " " & operacion & " " & valor2 & "
= " & resultado) '
    End Sub
    ' (Se asume que los botones numéricos se gestionan de forma similar al
Programa 128 para txtPantalla.Text)
End Class
```

- **Explicación:**
  - Las variables `valor1` y `operacion` se utilizan como en una calculadora básica.
  - El evento `btnOperacion_Click` almacena el primer valor y la operación.
  - Cuando se presiona `btnIgual`, se calcula el `resultado` de la operación entre `valor1` y `valor2` (obtenido de `txtPantalla`).
  - El `resultado` se muestra en `txtPantalla`, y además, se añade una cadena que describe la operación completa y su resultado (`valor1 & " " & operacion & " " & valor2 & " = " & resultado`) a `lstHistorial`.

Programa 133. Calculadora con memoria de resultados

Este programa añade funciones de memoria a la calculadora, permitiendo al usuario guardar, recuperar y borrar un valor en memoria.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtPantalla`.
  - `Button`: **Name:** `btnMS`, **Text:** "MS" (Memory Store).
  - `Button`: **Name:** `btnMR`, **Text:** "MR" (Memory Recall).
  - `Button`: **Name:** `btnMC`, **Text:** "MC" (Memory Clear).
- **Código VB.NET:**

```
Public Class Form1
    Dim memoria As Double = 0 '
    Private Sub btnMS_Click(sender As Object, e As EventArgs) Handles
btnMS.Click
        memoria = Val(txtPantalla.Text) '
    End Sub
    Private Sub btnMR_Click(sender As Object, e As EventArgs) Handles
btnMR.Click
        txtPantalla.Text = memoria '
    End Sub
    Private Sub btnMC_Click(sender As Object, e As EventArgs) Handles
btnMC.Click
        memoria = 0 '
    End Sub
End Class
```

- **Explicación:**
  - Se declara una variable `memoria` de tipo `Double` e inicializada a 0, para almacenar el valor en memoria.
  - Cuando se hace clic en `btnMS`, el valor numérico actual de `txtPantalla` se guarda en `memoria`.
  - Al hacer clic en `btnMR`, el valor almacenado en `memoria` se muestra en `txtPantalla`.
  - Cuando se hace clic en `btnMC`, la variable `memoria` se reinicia a 0.

Programa 134. Juego de preguntas con puntuación

Este programa implementa un juego de preguntas y respuestas que lleva un seguimiento de la puntuación del jugador.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblPregunta`.
  - `TextBox`: **Name:** `txtRespuesta`.
  - `Button`: **Name:** `btnVerificar`, **Text:** "Verificar".
  - `Label`: **Name:** `lblPuntaje`.
- **Código VB.NET:**

```
Public Class Form1
    Dim preguntas As New List(Of String) From {"Capital de Francia?",
"5+5?"} '
    Dim respuestas As New List(Of String) From {"París", "10"} '
    Dim indice As Integer = 0 '
    Dim puntaje As Integer = 0 '
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        lblPregunta.Text = preguntas(indice) '
        lblPuntaje.Text = "Puntaje: " & puntaje '
    End Sub
    Private Sub btnVerificar_Click(sender As Object, e As EventArgs)
Handles btnVerificar.Click
        If txtRespuesta.Text.ToLower() = respuestas(indice).ToLower() Then
puntaje += 1 '
        indice += 1 '
        If indice < preguntas.Count Then '
            lblPregunta.Text = preguntas(indice) '
            txtRespuesta.Clear() '
        Else
            MessageBox.Show("Juego terminado. Puntaje: " & puntaje) '
        End If
        lblPuntaje.Text = "Puntaje: " & puntaje '
    End Sub
End Class
```

- **Explicación:**
  - Las listas `preguntas` y `respuestas` almacenan el contenido del juego. `indice` sigue la pregunta actual, y `puntaje` la puntuación.
  - En `Form1_Load`, se muestra la primera pregunta y el puntaje inicial.
  - Cuando se hace clic en `btnVerificar`, el texto de `txtRespuesta` se compara (ignorando mayúsculas/minúsculas) con la respuesta correcta. Si es correcta, `puntaje` se incrementa.
  - `indice` avanza a la siguiente pregunta. Si quedan preguntas, se muestra la siguiente; si no, el juego termina y se muestra el puntaje final. `lblPuntaje` se actualiza con el puntaje actual.

Programa 135. Juego de preguntas con tiempo límite

Este programa es un juego de preguntas que incorpora un temporizador de cuenta regresiva, y el juego finaliza cuando el tiempo se agota.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblPregunta`.
  - `TextBox`: **Name:** `txtRespuesta`.
  - `Button`: **Name:** `btnVerificar`, **Text:** "Verificar".
  - `Label`: **Name:** `lblTiempo`.
  - `Timer`: **Name:** `Timer1`, **Interval:** `1000` (milisegundos).
- **Código VB.NET:**

```
Public Class Form1
    Dim tiempo As Integer = 10 '
    Dim pregunta As String = "Capital de España?" '
    Dim respuestaCorrecta As String = "Madrid" '
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        lblPregunta.Text = pregunta '
        lblTiempo.Text = "Tiempo: " & tiempo '
        Timer1.Start() '
    End Sub
    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
        tiempo -= 1 '
        lblTiempo.Text = "Tiempo: " & tiempo '
        If tiempo = 0 Then '
            Timer1.Stop() '
            MessageBox.Show("Se acabó el tiempo") '
            ' (Opcional: aquí se podría verificar la respuesta dada antes
de que el tiempo se acabe)
        End If
    End Sub
    ' (Opcional: el botón btnVerificar podría verificar la respuesta y
reiniciar el tiempo si es correcto, para nuevas preguntas)
End Class
```

- **Explicación:**
  - `tiempo` almacena los segundos restantes, y `pregunta` y `respuestaCorrecta` el contenido de la pregunta.
  - En `Form1_Load`, se muestra la `pregunta`, el `tiempo` inicial, y se inicia `Timer1`.
  - En cada `Timer1_Tick` (cada segundo), `tiempo` se decrementa y `lblTiempo` se actualiza.
  - Cuando `tiempo` llega a 0, `Timer1` se detiene y un `MessageBox` anuncia que el tiempo se ha agotado.

Programa 136. Juego de preguntas con niveles

Este programa demuestra cómo adaptar las preguntas según un nivel de dificultad seleccionado por el usuario en un `ComboBox`.

- **Controles necesarios en el formulario:**
    - `Label`: **Name:** `lblPregunta`.
    - `TextBox`: **Name:** `txtRespuesta`.
    - `Button`: **Name:** `btnVerificar`, **Text:** "Verificar".
    - `ComboBox`: **Name:** `cmbNivel`, **Items:** "Fácil", "Medio", "Difícil".
- **Código VB.NET:**

```
Public Class Form1
    Dim preguntasFacil As New List(Of String) From {"2+2?"} '
    Dim respuestasFacil As New List(Of String) From {"4"} '
    ' (Opcional: agregar listas para preguntasMedio y preguntasDificil)
    Private Sub cmbNivel_SelectedIndexChanged(sender As Object, e As
EventArgs) Handles cmbNivel.SelectedIndexChanged
        If cmbNivel.SelectedItem = "Fácil" Then '
            lblPregunta.Text = preguntasFacil(0) '
            ' (Opcional: cargar otras preguntas según el nivel
seleccionado)
        End If
    End Sub
    ' (Opcional: lógica para btnVerificar para comprobar respuestas y
avanzar)
End Class
```

- **Explicación:**
    - Se definen listas de preguntas y respuestas para cada nivel (en el ejemplo, solo `preguntasFacil` y `respuestasFacil`).
    - El evento `cmbNivel_SelectedIndexChanged` se dispara cuando el usuario selecciona un nuevo nivel. Si el nivel es "Fácil", `lblPregunta.Text` se actualiza con la primera pregunta de la lista `preguntasFacil`.
    - Para una implementación completa, se necesitarían listas de preguntas para "Medio" y "Difícil", y la lógica de `btnVerificar` para comprobar la respuesta y avanzar por las preguntas del nivel actual.

Programa 137. Juego de preguntas con base de datos

Este programa utiliza una base de datos SQLite para almacenar y recuperar preguntas y sus respuestas, integrando la persistencia de datos en un juego de preguntas.

- **Controles necesarios en el formulario:**
    - `Label`: **Name:** `lblPregunta`.
    - `TextBox`: **Name:** `txtRespuesta`.
    - `Button`: **Name:** `btnVerificar`, **Text:** "Verificar".
- **Requisito adicional:**
    - Requiere agregar una referencia a la librería **System.Data.SQLite** (generalmente a través de NuGet).
- **Código VB.NET:**

```
Imports System.Data.SQLite '
Public Class Form1
    Dim conn As New SQLiteConnection("Data
Source=preguntas.db;Version=3;") '
    Dim cmd As SQLiteCommand '
    Dim dr As SQLiteDataReader '
    Dim preguntaActual As String = "" '
    Dim respuestaActual As String = "" '
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        conn.Open() '
        cmd = New SQLiteCommand("CREATE TABLE IF NOT EXISTS
preguntas(pregunta TEXT, respuesta TEXT)", conn) '
        cmd.ExecuteNonQuery() '
        ' Opcional: Insertar algunas preguntas si la tabla está vacía
        ' cmd = New SQLiteCommand("INSERT INTO preguntas (pregunta,
respuesta) VALUES ('¿Cuál es la capital de España?', 'Madrid')", conn)
        ' cmd.ExecuteNonQuery()
        cmd = New SQLiteCommand("SELECT * FROM preguntas LIMIT 1", conn) '
        dr = cmd.ExecuteReader() '
        If dr.Read() Then '
            preguntaActual = dr("pregunta") '
            respuestaActual = dr("respuesta") '
            lblPregunta.Text = preguntaActual '
        End If
        dr.Close() '
    End Sub
    Private Sub btnVerificar_Click(sender As Object, e As EventArgs)
Handles btnVerificar.Click
        If txtRespuesta.Text.ToLower() = respuestaActual.ToLower() Then '
            MessageBox.Show("Correcto") '
        Else
            MessageBox.Show("Incorrecto") '
        End If
        ' (Opcional: cargar la siguiente pregunta de la base de datos)
    End Sub
End Class
```

- **Explicación:**
    - Se importa `System.Data.SQLite` y se establece una conexión (`conn`) a un archivo de base de datos `preguntas.db`.

- En `Form1_Load`, la conexión se abre. Si la tabla `preguntas` no existe, se crea. Luego, se ejecuta una consulta `SELECT` para obtener la primera pregunta y respuesta de la base de datos, que se almacenan en `preguntaActual` y `respuestaActual` y se muestran en `lblPregunta`. El `DataReader` se cierra.
- Cuando se hace clic en `btnVerificar`, el texto del `txtRespuesta` se compara con `respuestaActual` (ignorando mayúsculas/minúsculas), y se muestra un `MessageBox` indicando si es "Correcto" o "Incorrecto".

Programa 138. Juego de preguntas con imágenes

Este programa implementa un juego de preguntas donde la pregunta se presenta visualmente a través de una imagen.

- **Controles necesarios en el formulario:**
    - `PictureBox`: **Name:** `picPregunta`.
    - `TextBox`: **Name:** `txtRespuesta`.
    - `Button`: **Name:** `btnVerificar`, **Text:** "Verificar".
- **Recursos adicionales:**
    - Se requiere tener archivos de imagen (ej. `imagen1.jpg`) en la misma carpeta que el ejecutable del programa o especificar una ruta completa.
- **Código VB.NET:**

```
Public Class Form1
    Dim preguntas As New List(Of String) From {"imagen1.jpg"} '
    Dim respuestas As New List(Of String) From {"Gato"} '
    Dim indice As Integer = 0 '
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        picPregunta.Image = Image.FromFile(preguntas(indice)) '
    End Sub
    Private Sub btnVerificar_Click(sender As Object, e As EventArgs)
Handles btnVerificar.Click
        If txtRespuesta.Text.ToLower() = respuestas(indice).ToLower() Then
'
            MessageBox.Show("Correcto") '
        Else
            MessageBox.Show("Incorrecto") '
        End If
        ' (Opcional: avanzar al siguiente índice y cargar la próxima
imagen y respuesta)
    End Sub
End Class
```

- **Explicación:**
    - Las listas `preguntas` (que contiene nombres de archivos de imagen) y `respuestas` almacenan el contenido del juego. `indice` sigue la pregunta actual.
    - En `Form1_Load`, la imagen correspondiente al `indice` actual se carga en `picPregunta` usando `Image.FromFile()`.
    - Cuando se hace clic en `btnVerificar`, la respuesta ingresada en `txtRespuesta` se compara (sin distinguir mayúsculas/minúsculas) con la respuesta correcta asociada a la imagen actual. Se muestra un `MessageBox` con el resultado.

Programa 139. Juego de preguntas con sonidos

Este programa implementa un juego de preguntas donde la pista para la respuesta es un sonido, que el usuario puede reproducir.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnReproducir`, **Text:** "Reproducir Sonido".
  - `TextBox`: **Name:** `txtRespuesta`.
  - `Button`: **Name:** `btnVerificar`, **Text:** "Verificar".
- **Recursos adicionales:**
  - Se requiere tener un archivo de sonido (ej. `sonido1.wav`) en la misma carpeta que el ejecutable del programa.
- **Código VB.NET:**

```
Imports System.Media '
Public Class Form1
    Dim sonido As String = "sonido1.wav" '
    Dim respuestaCorrecta As String = "Campana" '
    Private Sub btnReproducir_Click(sender As Object, e As EventArgs)
Handles btnReproducir.Click
        Dim player As New SoundPlayer(sonido) '
        player.Play() '
    End Sub
    Private Sub btnVerificar_Click(sender As Object, e As EventArgs)
Handles btnVerificar.Click
        If txtRespuesta.Text.ToLower() = respuestaCorrecta.ToLower() Then
'
            MessageBox.Show("Correcto") '
        Else
            MessageBox.Show("Incorrecto") '
        End If
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.Media` para la reproducción de audio. Las variables `sonido` y `respuestaCorrecta` almacenan la ruta del archivo de sonido y la respuesta esperada.
  - Cuando se hace clic en `btnReproducir`, se crea una nueva instancia de `SoundPlayer` con el archivo de sonido y se reproduce con `player.Play()`.
  - Al hacer clic en `btnVerificar`, la respuesta ingresada en `txtRespuesta` se compara (ignorando mayúsculas/minúsculas) con `respuestaCorrecta`, y se muestra un `MessageBox` con el resultado.

Programa 140. Juego de preguntas multijugador

Este programa simula un juego de preguntas para dos jugadores, llevando la puntuación de cada uno.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblPregunta`.
  - `TextBox`: **Name:** `txtRespuesta1` (para Jugador 1).
  - `TextBox`: **Name:** `txtRespuesta2` (para Jugador 2).
  - `Button`: **Name:** `btnVerificar`, **Text:** "Verificar".
  - `Label`: **Name:** `lblPuntaje` (para mostrar las puntuaciones).
- **Código VB.NET:**

```
Public Class Form1
    Dim preguntas As New List(Of String) From {"Capital de Italia?"} '
    Dim respuestas As New List(Of String) From {"Roma"} '
    Dim puntaje1 As Integer = 0 '
    Dim puntaje2 As Integer = 0 '
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        lblPregunta.Text = preguntas(0) '
        lblPuntaje.Text = "Jugador1: 0 - Jugador2: 0" '
    End Sub
    Private Sub btnVerificar_Click(sender As Object, e As EventArgs)
Handles btnVerificar.Click
        If txtRespuesta1.Text.ToLower() = respuestas(0).ToLower() Then
puntaje1 += 1 '
        If txtRespuesta2.Text.ToLower() = respuestas(0).ToLower() Then
puntaje2 += 1 '
        lblPuntaje.Text = "Jugador1: " & puntaje1 & " - Jugador2: " &
puntaje2 '
        ' (Opcional: avanzar a la siguiente pregunta si hay más en las
listas)
    End Sub
End Class
```

- **Explicación:**
  - Las listas `preguntas` y `respuestas` almacenan el contenido del juego. `puntaje1` y `puntaje2` llevan la puntuación de cada jugador.
  - En `Form1_Load`, se establece la primera pregunta y se inicializa `lblPuntaje`.
  - Al hacer clic en `btnVerificar`, las respuestas de ambos jugadores (`txtRespuesta1.Text` y `txtRespuesta2.Text`) se comparan con la respuesta correcta. Si son correctas, se incrementa el puntaje correspondiente.
  - `lblPuntaje` se actualiza para mostrar las puntuaciones actuales de ambos jugadores.

Programa 141. Juego de preguntas con ranking

Este programa implementa un juego de preguntas que lleva un registro del puntaje actual del jugador y lo muestra en un ranking.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblPregunta`.
  - `TextBox`: **Name:** `txtRespuesta`.
  - `Button`: **Name:** `btnVerificar`, **Text:** "Verificar".
  - `ListBox`: **Name:** `lstRanking`.
- **Código VB.NET:**

```
Public Class Form1
    Dim preguntas As New List(Of String) From {"Capital de Alemania?"} '
    Dim respuestas As New List(Of String) From {"Berlín"} '
    Dim puntajes As New List(Of Integer) '
    Dim indice As Integer = 0 '
    Dim puntajeActual As Integer = 0 '
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        lblPregunta.Text = preguntas(indice) '
    End Sub
    Private Sub btnVerificar_Click(sender As Object, e As EventArgs)
Handles btnVerificar.Click
        If txtRespuesta.Text.ToLower() = respuestas(indice).ToLower() Then
puntajeActual += 1 '
        puntajes.Add(puntajeActual) '
        lstRanking.Items.Clear() '
        For Each p In puntajes '
            lstRanking.Items.Add("Puntaje: " & p) '
        Next
        indice += 1 '
        If indice < preguntas.Count Then lblPregunta.Text =
preguntas(indice) '
        ' (Opcional: manejar el fin del juego si no hay más preguntas)
    End Sub
End Class
```

- **Explicación:**
  - Las listas `preguntas` y `respuestas` contienen el contenido del juego. `puntajes` almacena los puntajes de las rondas (o el `puntajeActual` acumulado). `indice` y `puntajeActual` siguen el progreso.
  - En `Form1_Load`, se muestra la primera pregunta.
  - Al hacer clic en `btnVerificar`, se comprueba la respuesta. Si es correcta, `puntajeActual` se incrementa.
  - El `puntajeActual` se añade a la lista `puntajes`. `lstRanking` se limpia y se repuebla con todos los puntajes registrados hasta el momento.
  - El `indice` avanza, y si hay más preguntas, se muestra la siguiente.

Programa 142. Juego de preguntas con exportación de resultados

Este programa permite exportar los resultados de un juego de preguntas, mostrados en un `ListBox`, a un archivo de texto.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnExportar`, **Text:** "Exportar Resultados".
  - `ListBox`: **Name:** `lstResultados`.
- **Código VB.NET:**

```
Imports System.IO '
Public Class Form1
    Private Sub btnExportar_Click(sender As Object, e As EventArgs)
Handles btnExportar.Click
        Dim archivo As String = "resultados.txt" '
        File.WriteAllLines(archivo, lstResultados.Items.Cast(Of String)())
'
        MessageBox.Show("Resultados exportados a " & archivo) '
    End Sub
End Class
```

- **Explicación:**
  - Se importa el espacio de nombres `System.IO` para operaciones de archivo.
  - Cuando se hace clic en `btnExportar`, se define un nombre de archivo (`"resultados.txt"`).
  - `File.WriteAllLines()` escribe todos los elementos del `lstResultados` como líneas separadas en el archivo especificado. `Cast(Of String)()` es necesario para convertir la colección `Items` a una colección de cadenas.
  - Finalmente, un `MessageBox` confirma que los resultados se han exportado.

Programa 143. Juego de preguntas con importación de resultados

Este programa permite importar resultados de un juego de preguntas desde un archivo de texto y mostrarlos en un `ListBox`.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnImportar`, **Text:** "Importar Resultados".
  - `ListBox`: **Name:** `lstResultados`.
  - `OpenFileDialog`: **Name:** `OpenFileDialog1`.
- **Código VB.NET:**

```
Imports System.IO '
Public Class Form1
    Private Sub btnImportar_Click(sender As Object, e As EventArgs)
Handles btnImportar.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then '
            lstResultados.Items.Clear() '

lstResultados.Items.AddRange(File.ReadAllLines(OpenFileDialog1.FileName))
'
        End If
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.IO`.
  - Al hacer clic en `btnImportar`, `OpenFileDialog1.ShowDialog()` abre un cuadro de diálogo para que el usuario seleccione un archivo de texto.
  - Si el usuario selecciona un archivo y hace clic en "Abrir" (`DialogResult.OK`), `lstResultados` se limpia.
  - `File.ReadAllLines(OpenFileDialog1.FileName)` lee todas las líneas del archivo seleccionado en una matriz de cadenas, y `lstResultados.Items.AddRange()` añade estas líneas al `ListBox`.

Programa 144. Juego de preguntas con estadísticas

Este programa muestra estadísticas en tiempo real del juego de preguntas, como el número de respuestas correctas y el total de intentos.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblPregunta`.
  - `TextBox`: **Name:** `txtRespuesta`.
  - `Button`: **Name:** `btnVerificar`, **Text:** "Verificar".
  - `Label`: **Name:** `lblEstadisticas`.
- **Código VB.NET:**

```
Public Class Form1
    Dim preguntas As New List(Of String) From {"2+2?", "Capital de
Italia?"} '
    Dim respuestas As New List(Of String) From {"4", "Roma"} '
    Dim correctas As Integer = 0 '
    Dim intentos As Integer = 0 '
    Dim indice As Integer = 0 '
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        lblPregunta.Text = preguntas(indice) '
    End Sub
    Private Sub btnVerificar_Click(sender As Object, e As EventArgs)
Handles btnVerificar.Click
        intentos += 1 '
        If txtRespuesta.Text.ToLower() = respuestas(indice).ToLower() Then
correctas += 1 '
        indice += 1 '
        If indice < preguntas.Count Then '
            lblPregunta.Text = preguntas(indice) '
            txtRespuesta.Clear() ' (no en la fuente, pero buena práctica)
        Else
            lblPregunta.Text = "Juego terminado" '
        End If
        lblEstadisticas.Text = "Correctas: " & correctas & " / Intentos: "
& intentos '
    End Sub
End Class
```

- **Explicación:**
  - Las variables `correctas` y `intentos` almacenan la cantidad de respuestas correctas y el total de intentos, respectivamente. `indice` sigue la pregunta actual.
  - En `Form1_Load`, se muestra la primera pregunta.
  - Al hacer clic en `btnVerificar`, `intentos` se incrementa. Si la respuesta es correcta, `correctas` también se incrementa.
  - Luego, `indice` avanza. Si hay más preguntas, se muestra la siguiente; si no, `lblPregunta` indica que el juego ha terminado.
  - Finalmente, `lblEstadisticas` se actualiza para mostrar el recuento de respuestas correctas y el total de intentos.

Programa 145. Juego de preguntas con gráficos

Este programa visualiza los resultados de un juego de preguntas utilizando un control `Chart` para mostrar un gráfico de columnas.

- **Controles necesarios en el formulario:**
  - `Chart`: **Name:** `chartResultados`.
  - `Button`: **Name:** `btnActualizar`, **Text:** "Actualizar Gráfico".
- **Requisito adicional:**
  - Asegurarse de que el control `Chart` esté disponible en el `Toolbox` (puede requerir añadir la referencia `System.Windows.Forms.DataVisualization` al proyecto).
- **Código VB.NET:**

```
Imports System.Windows.Forms.DataVisualization.Charting '
Public Class Form1
    Dim correctas As Integer = 3 '
    Dim incorrectas As Integer = 2 '
    Private Sub btnActualizar_Click(sender As Object, e As EventArgs)
Handles btnActualizar.Click
        chartResultados.Series.Clear() '
        chartResultados.Series.Add("Resultados") '
        chartResultados.Series("Resultados").Points.AddXY("Correctas",
correctas) '
        chartResultados.Series("Resultados").Points.AddXY("Incorrectas",
incorrectas) '
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.Windows.Forms.DataVisualization.Charting` para usar el control `Chart`. Las variables `correctas` e `incorrectas` (`3` y `2` respectivamente en el ejemplo) representan los datos a graficar.
  - Cuando se hace clic en `btnActualizar`, `chartResultados.Series.Clear()` borra cualquier serie de datos existente en el gráfico.
  - Se añade una nueva serie llamada "Resultados".
  - `chartResultados.Series("Resultados").Points.AddXY()` añade dos puntos de datos: uno para "Correctas" con el valor de `correctas`, y otro para "Incorrectas" con el valor de `incorrectas`. Esto crea un gráfico de columnas que visualiza estos dos valores.

Programa 146. Juego de preguntas con personalización

Este programa permite al usuario añadir sus propias preguntas a una lista, personalizando el contenido del juego.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtPregunta`.
    - `Button`: **Name:** `btnAgregarPregunta`, **Text:** "Agregar Pregunta".
    - `ListBox`: **Name:** `lstPreguntas`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnAgregarPregunta_Click(sender As Object, e As EventArgs)
Handles btnAgregarPregunta.Click
        lstPreguntas.Items.Add(txtPregunta.Text) '
        txtPregunta.Clear() ' (no en la fuente, pero buena práctica)
    End Sub
End Class
```

- **Explicación:**
    - Al hacer clic en `btnAgregarPregunta`, el texto que el usuario ha introducido en `txtPregunta` se añade como un nuevo elemento a `lstPreguntas`. Esto permite al usuario ver las preguntas que ha añadido. Para un juego completo, también se necesitaría un `TextBox` para la respuesta y un mecanismo para almacenar ambas.

Programa 147. Juego de preguntas con guardado automático

Este programa guarda automáticamente las respuestas del usuario en un archivo de texto cada cierto tiempo, además de un guardado manual.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtRespuesta`.
    - `Button`: **Name:** `btnGuardar`, **Text:** "Guardar".
    - `Timer`: **Name:** `Timer1`, **Interval:** `5000` (milisegundos).
- **Código VB.NET:**

```
Imports System.IO '
Public Class Form1
    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
        File.AppendAllText("respuestas.txt", txtRespuesta.Text &
Environment.NewLine) '
        ' (Opcional: limpiar txtRespuesta o mostrar un mensaje de
guardado)
    End Sub
    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click
        File.AppendAllText("respuestas.txt", txtRespuesta.Text &
Environment.NewLine) '
        MessageBox.Show("Respuesta guardada manualmente.") ' (no en la
fuente, pero útil)
    End Sub
    ' (Opcional: iniciar Timer1 en Form_Load)
End Class
```

- **Explicación:**
    - Se importa `System.IO` para la manipulación de archivos.
    - El `Timer1` (que debería iniciarse en `Form_Load` y está configurado para dispararse cada 5 segundos) llama a `File.AppendAllText()` en su evento `Timer1_Tick` para añadir el contenido actual de `txtRespuesta` (seguido de un salto de línea) al archivo "respuestas.txt".
    - El botón `btnGuardar` realiza la misma acción de guardar la respuesta manualmente al hacer clic.

Programa 148. Juego de preguntas con recuperación de partida

Este programa permite guardar el progreso de un juego de preguntas y recuperarlo posteriormente, cargando la pregunta correcta.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnGuardar`, **Text:** "Guardar Partida".
  - `Button`: **Name:** `btnCargar`, **Text:** "Cargar Partida".
  - `Label`: **Name:** `lblPregunta`.
- **Código VB.NET:**

```
Imports System.IO '
Public Class Form1
    Dim preguntas As New List(Of String) From {"Capital de Francia?",
"2+2?"} '
    Dim indice As Integer = 0 '
    ' (Opcional: inicializar lblPregunta.Text en Form_Load con
preguntas(indice))
    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click
        File.WriteAllText("partida.txt", indice.ToString()) '
        MessageBox.Show("Partida guardada.") ' (no en la fuente, pero
útil)
    End Sub
    Private Sub btnCargar_Click(sender As Object, e As EventArgs) Handles
btnCargar.Click
        If File.Exists("partida.txt") Then '
            indice = CInt(File.ReadAllText("partida.txt")) '
            lblPregunta.Text = preguntas(indice) '
            MessageBox.Show("Partida cargada.") ' (no en la fuente, pero
útil)
        Else
            MessageBox.Show("No hay partida guardada.") ' (no en la
fuente, pero útil)
        End If
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.IO`. Las variables `preguntas` (lista de preguntas) e `indice` (posición de la pregunta actual) almacenan el estado del juego.
  - Cuando se hace clic en `btnGuardar`, el valor actual de `indice` se escribe en un archivo de texto llamado "partida.txt".
  - Al hacer clic en `btnCargar`, el programa verifica si "partida.txt" existe. Si es así, lee el `indice` guardado desde el archivo, lo convierte a entero (`CInt()`), y actualiza `lblPregunta.Text` con la pregunta correspondiente de la lista `preguntas`.

Programa 149. Juego de preguntas con dificultad progresiva

Este programa implementa un juego de preguntas que aumenta su dificultad a medida que el jugador avanza de nivel.

- **Controles necesarios en el formulario:**
    - `Label`: **Name:** `lblPregunta`.
    - `TextBox`: **Name:** `txtRespuesta`.
    - `Button`: **Name:** `btnVerificar`, **Text:** "Verificar".
- **Código VB.NET:**

```
Public Class Form1
    Dim preguntasFacil As New List(Of String) From {"2+2?"} '
    Dim preguntasMedio As New List(Of String) From {"Capital de España?"}
'
    Dim preguntasDificil As New List(Of String) From {"Raíz cuadrada de
256?"} '
    Dim nivel As Integer = 1 '
    ' (Opcional: Listas de respuestas para cada nivel, como en otros
ejemplos de juego de preguntas)
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        lblPregunta.Text = preguntasFacil(0) '
    End Sub
    Private Sub btnVerificar_Click(sender As Object, e As EventArgs)
Handles btnVerificar.Click
        ' (Opcional: Aquí se debería verificar la respuesta actual antes
de cambiar de nivel)
        Select Case nivel '
            Case 1 : lblPregunta.Text = preguntasMedio(0) : nivel += 1 '
            Case 2 : lblPregunta.Text = preguntasDificil(0) : nivel += 1 '
            Case 3 : lblPregunta.Text = "Juego terminado" '
        End Select
        txtRespuesta.Clear() ' (no en la fuente, pero buena práctica)
    End Sub
End Class
```

- **Explicación:**
    - Se definen listas de preguntas para los niveles "Fácil", "Medio" y "Difícil" (`preguntasFacil`, `preguntasMedio`, `preguntasDificil`). La variable `nivel` controla la dificultad actual (1 para Fácil, 2 para Medio, etc.).
    - En `Form1_Load`, el juego comienza mostrando la primera pregunta del nivel "Fácil".
    - Cuando se hace clic en `btnVerificar`, un `Select Case` evalúa el `nivel` actual. Después de (opcionalmente) verificar la respuesta, el `lblPregunta.Text` se actualiza con la primera pregunta del siguiente nivel, y la variable `nivel` se incrementa.
    -
    - Una vez que se llega al `Case 3`, se indica que el juego ha terminado.

Programa 150. Juego de preguntas con logros desbloqueables

Este programa implementa un juego de preguntas que recompensa al jugador con "logros" al alcanzar ciertos hitos de puntuación.

- **Controles necesarios en el formulario:**
  - `Label`: **Name:** `lblPregunta`.
  - `TextBox`: **Name:** `txtRespuesta`.
  - `Button`: **Name:** `btnVerificar`, **Text:** "Verificar".
  - `ListBox`: **Name:** `lstLogros`.
- **Código VB.NET:**

```
Public Class Form1
    Dim preguntas As New List(Of String) From {"2+2?", "Capital de
Francia?"} '
    Dim respuestas As New List(Of String) From {"4", "París"} '
    Dim puntaje As Integer = 0 '
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        lblPregunta.Text = preguntas(0) '
    End Sub
    Private Sub btnVerificar_Click(sender As Object, e As EventArgs)
Handles btnVerificar.Click
        Dim indice As Integer = lstLogros.Items.Count ' (este índice
podría ser para la pregunta actual, no logros)
        ' Asumiendo que 'indice' debería ser un campo de clase para la
pregunta actual:
        ' If txtRespuesta.Text.ToLower() =
respuestas(currentQuestionIndex).ToLower() Then puntaje += 1
        ' La fuente usa lstLogros.Items.Count como indice, lo cual es
peculiar si se refiere a la pregunta.
        ' Para este ejemplo, interpretamos que el indice aquí es un
contador de preguntas o una forma de avanzar.
        If txtRespuesta.Text.ToLower() =
respuestas(lstLogros.Items.Count).ToLower() Then puntaje += 1 '

        If puntaje = 1 Then lstLogros.Items.Add("Primer acierto!") '
        If puntaje = 2 Then lstLogros.Items.Add("Dos aciertos seguidos!")
'
        ' (Opcional: avanzar a la siguiente pregunta y limpiar
txtRespuesta)
    End Sub
End Class
```

- **Explicación:**
  - Las listas `preguntas` y `respuestas` contienen el contenido del juego. `puntaje` lleva la puntuación del jugador.
  - En `Form1_Load`, se muestra la primera pregunta.
  - Cuando se hace clic en `btnVerificar`, el texto de `txtRespuesta` se compara (ignorando mayúsculas/minúsculas) con la respuesta correcta. Si es correcta, `puntaje` se incrementa.
  - Después de actualizar el `puntaje`, se verifica si el `puntaje` ha alcanzado ciertos umbrales. Si `puntaje` es 1, se añade el logro "Primer acierto!" a `lstLogros`. Si `puntaje` es 2, se añade el logro "Dos aciertos seguidos!" a `lstLogros`.

- **Nota:** La implementación del `indice` en la línea `If txtRespuesta.Text.ToLower() = respuestas(indice).ToLower()` usando `lstLogros.Items.Count` como `indice` es inusual. Normalmente, `indice` sería una variable de clase que rastrea la pregunta actual, independientemente de los logros. Para que este programa sea un juego de preguntas completo, se necesitaría gestionar el avance de las preguntas de forma independiente de la cantidad de logros.

Programa 151. Editor de imágenes simple

Este programa permite cargar una imagen y aplicarle un filtro básico de escala de grises.

- **Controles necesarios en el formulario:**
  - `PictureBox`: **Name:** `picImagen`, **SizeMode:** `StretchImage`.
  - `Button`: **Name:** `btnCargar`, **Text:** "Cargar Imagen".
  - `Button`: **Name:** `btnGrises`, **Text:** "Convertir a Gris".
  - `OpenFileDialog`: **Name:** `OpenFileDialog1`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnCargar_Click(sender As Object, e As EventArgs) Handles
btnCargar.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            picImagen.Image = Image.FromFile(OpenFileDialog1.FileName)
        End If
    End Sub
    Private Sub btnGrises_Click(sender As Object, e As EventArgs) Handles
btnGrises.Click
        If picImagen.Image IsNot Nothing Then
            Dim bmp As New Bitmap(picImagen.Image)
            For x As Integer = 0 To bmp.Width - 1
                For y As Integer = 0 To bmp.Height - 1
                    Dim c As Color = bmp.GetPixel(x, y)
                    Dim gris As Integer = CInt((c.R + c.G + c.B) / 3)
                    bmp.SetPixel(x, y, Color.FromArgb(gris, gris, gris))
                Next
            Next
            picImagen.Image = bmp
        End If
    End Sub
End Class
```

- **Explicación:**
  - **Cargar Imagen:** Cuando se hace clic en `btnCargar`, el `OpenFileDialog1` se abre para que el usuario seleccione un archivo de imagen. Si se selecciona, la imagen se carga en `picImagen.Image`.
  - **Convertir a Gris:** El evento `btnGrises_Click` se activa para convertir la imagen a escala de grises. Primero, verifica que haya una imagen cargada. Luego, crea una copia `Bitmap` de la imagen. Un doble bucle `For` itera sobre cada píxel de la imagen. Para cada píxel, obtiene su color (`c`), calcula un valor de gris promediando los componentes Rojo, Verde y Azul (`(c.R + c.G + c.B) / 3`), y establece el píxel con el nuevo color gris (`Color.FromArgb(gris, gris, gris)`). Finalmente, la imagen modificada (`bmp`) se asigna de nuevo a `picImagen.Image`.

Programa 152. Editor de audio simple

Este programa permite cargar un archivo de audio y reproducirlo.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnCargar`, **Text:** "Cargar Audio".
  - `Button`: **Name:** `btnReproducir`, **Text:** "Reproducir".
  - `OpenFileDialog`: **Name:** `OpenFileDialog1`.
- **Código VB.NET:**

```
Imports System.Media
Public Class Form1
    Dim player As SoundPlayer
    Private Sub btnCargar_Click(sender As Object, e As EventArgs) Handles
btnCargar.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            player = New SoundPlayer(OpenFileDialog1.FileName)
        End If
    End Sub
    Private Sub btnReproducir_Click(sender As Object, e As EventArgs)
Handles btnReproducir.Click
        If player IsNot Nothing Then player.Play()
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.Media` para utilizar la clase `SoundPlayer`. Una instancia de `SoundPlayer` llamada `player` se declara a nivel de clase.
  - **Cargar Audio:** Al hacer clic en `btnCargar`, se abre `OpenFileDialog1`. Si el usuario selecciona un archivo, `player` se inicializa con la ruta del archivo de audio elegido.
  - **Reproducir:** Al hacer clic en `btnReproducir`, si `player` no es nulo (es decir, si se ha cargado un archivo), el método `player.Play()` reproduce el sonido.

Programa 153. Reproductor de música básico

Este programa implementa un reproductor de música con funciones de cargar, reproducir y detener, utilizando el control `AxWindowsMediaPlayer`.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnCargar`, **Text:** "Cargar Música".
  - `Button`: **Name:** `btnPlay`, **Text:** "Play".
  - `Button`: **Name:** `btnStop`, **Text:** "Stop".
  - `OpenFileDialog`: **Name:** `OpenFileDialog1`.
  - `AxWindowsMediaPlayer`: **Name:** `wmp`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnCargar_Click(sender As Object, e As EventArgs) Handles
btnCargar.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            wmp.URL = OpenFileDialog1.FileName
        End If
    End Sub
    Private Sub btnPlay_Click(sender As Object, e As EventArgs) Handles
btnPlay.Click
        wmp.Ctlcontrols.play()
    End Sub
    Private Sub btnStop_Click(sender As Object, e As EventArgs) Handles
btnStop.Click
        wmp.Ctlcontrols.stop()
    End Sub
End Class
```

- **Explicación:**
  - Este programa requiere que se añada el componente **AxWindowsMediaPlayer** al formulario (generalmente arrastrándolo desde el Cuadro de Herramientas después de agregar la referencia del control COM).
  - **Cargar Música:** Cuando se hace clic en `btnCargar`, se abre `OpenFileDialog1`. Si se selecciona un archivo, la propiedad `wmp.URL` se establece con la ruta del archivo elegido, lo que lo carga en el reproductor.
  - **Play:** Al hacer clic en `btnPlay`, `wmp.Ctlcontrols.play()` inicia la reproducción del archivo cargado.
  - **Stop:** Al hacer clic en `btnStop`, `wmp.Ctlcontrols.stop()` detiene la reproducción.

Programa 154. Reproductor de video básico

Este programa permite cargar y reproducir un archivo de video.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnCargar`, **Text:** "Cargar Video".
  - `AxWindowsMediaPlayer`: **Name:** `wmp`.
  - `OpenFileDialog`: **Name:** `OpenFileDialog1`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnCargar_Click(sender As Object, e As EventArgs) Handles
btnCargar.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            wmp.URL = OpenFileDialog1.FileName
            wmp.Ctlcontrols.play()
        End If
    End Sub
End Class
```

- **Explicación:**
  - Similar al reproductor de música, este programa utiliza el control `AxWindowsMediaPlayer`.
  - **Cargar Video:** Al hacer clic en `btnCargar`, se abre `OpenFileDialog1`. Si el usuario selecciona un archivo de video, su ruta se asigna a `wmp.URL`, y el video comienza a reproducirse automáticamente con `wmp.Ctlcontrols.play()`.

Programa 155. Reproductor multimedia integrado

Este programa combina la funcionalidad de los reproductores de audio y video, permitiendo cargar ambos tipos de archivos usando filtros específicos en el `OpenFileDialog`.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnCargarAudio`, **Text:** "Cargar Audio".
  - `Button`: **Name:** `btnCargarVideo`, **Text:** "Cargar Video".
  - `AxWindowsMediaPlayer`: **Name:** `wmp`.
  - `OpenFileDialog`: **Name:** `OpenFileDialog1`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnCargarAudio_Click(sender As Object, e As EventArgs)
Handles btnCargarAudio.Click
        OpenFileDialog1.Filter = "WAV|*.wav|MP3|*.mp3"
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            wmp.URL = OpenFileDialog1.FileName
        End If
    End Sub
    Private Sub btnCargarVideo_Click(sender As Object, e As EventArgs)
Handles btnCargarVideo.Click
        OpenFileDialog1.Filter = "MP4|*.mp4|AVI|*.avi"
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            wmp.URL = OpenFileDialog1.FileName
        End If
    End Sub
End Class
```

- **Explicación:**
  - Este programa también utiliza el control `AxWindowsMediaPlayer`.
  - **Cargar Audio:** Al hacer clic en `btnCargarAudio`, la propiedad `Filter` de `OpenFileDialog1` se establece para mostrar solo archivos WAV y MP3. Si se selecciona un archivo, su ruta se carga en `wmp.URL`.
  - **Cargar Video:** Al hacer clic en `btnCargarVideo`, el filtro de `OpenFileDialog1` se cambia para mostrar solo archivos MP4 y AVI. De manera similar, si se selecciona un archivo, su ruta se carga en `wmp.URL`. El `AxWindowsMediaPlayer` manejará la reproducción según el tipo de archivo.

Programa 156. Aplicación que controla volumen del sistema

Este programa muestra conceptualmente cómo se interactuaría con el volumen del sistema, aunque la implementación real requiere llamadas avanzadas a la API del sistema operativo.

- **Controles necesarios en el formulario:**
  - `TrackBar`: **Name:** `trkVolumen`, **Minimum:** `0`, **Maximum:** `100`.
  - `Label`: **Name:** `lblVolumen`.
- **Código VB.NET:**

```
Imports System.Runtime.InteropServices
Public Class Form1
    <DllImport("user32.dll")>
    Public Shared Function SendMessageW(ByVal hWnd As IntPtr, ByVal Msg As
Integer, ByVal wParam As Integer, ByVal lParam As Integer) As Integer
    End Function
    Private Sub trkVolumen_Scroll(sender As Object, e As EventArgs)
Handles trkVolumen.Scroll
        lblVolumen.Text = "Volumen: " & trkVolumen.Value
        ' Nota: Cambiar volumen global requiere API avanzada o bibliotecas
externas
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.Runtime.InteropServices` para permitir la interoperabilidad con funciones de la API de Windows (DLL).
  - Se declara una función `SendMessageW` con `DllImport`, que es una función genérica para enviar mensajes a ventanas, pero no se implementa el uso directo de esta función para cambiar el volumen del sistema en el código proporcionado.
  - El evento `trkVolumen_Scroll` simplemente actualiza un `lblVolumen` con el valor actual del `TrackBar`. La nota en el código aclara que el control real del volumen global es más complejo y requeriría el uso de APIs de sistema o bibliotecas externas.

Programa 157. Aplicación que controla brillo de pantalla

Similar al control de volumen, este programa muestra el concepto de ajustar el brillo de la pantalla, pero la implementación directa es compleja.

- **Controles necesarios en el formulario:**
  - `TrackBar`: **Name:** `trkBrillo`, **Minimum:** `0`, **Maximum:** `100`.
  - `Label`: **Name:** `lblBrillo`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub trkBrillo_Scroll(sender As Object, e As EventArgs) Handles
trkBrillo.Scroll
        lblBrillo.Text = "Brillo: " & trkBrillo.Value
        ' Nota: Ajustar brillo real requiere WMI o APIs específicas
    End Sub
End Class
```

- **Explicación:**
  - El evento `trkBrillo_Scroll` simplemente actualiza un `lblBrillo` con el valor actual del `TrackBar`. La nota en el código indica que ajustar el brillo real de la pantalla es una tarea que requiere tecnologías como WMI (Windows Management Instrumentation) o APIs específicas del sistema, lo cual no se implementa en este ejemplo.

Programa 158. Aplicación que toma capturas de pantalla

Este programa permite al usuario tomar una captura de pantalla de la pantalla principal y mostrarla en un `PictureBox`.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnCaptura`, **Text:** "Capturar Pantalla".
  - `PictureBox`: **Name:** `picPantalla`, **SizeMode:** `StretchImage`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnCaptura_Click(sender As Object, e As EventArgs) Handles
btnCaptura.Click
        Dim bmp As New Bitmap(Screen.PrimaryScreen.Bounds.Width,
Screen.PrimaryScreen.Bounds.Height)
        Dim g As Graphics = Graphics.FromImage(bmp)
        g.CopyFromScreen(0, 0, 0, 0, bmp.Size)
        picPantalla.Image = bmp
    End Sub
End Class
```

- **Explicación:**
  - Cuando se hace clic en `btnCaptura`, el evento `btnCaptura_Click` se ejecuta.
  - Se crea un nuevo objeto `Bitmap` (`bmp`) con las dimensiones de la pantalla principal (`Screen.PrimaryScreen.Bounds.Width`, `Screen.PrimaryScreen.Bounds.Height`).
  - Se obtiene un objeto `Graphics` (`g`) a partir de este `Bitmap`.
  - `g.CopyFromScreen(0, 0, 0, 0, bmp.Size)` copia el contenido de la pantalla a la imagen `bmp`. Los primeros dos `0, 0` son las coordenadas de origen en la pantalla, los segundos `0, 0` son las coordenadas de destino en el `Bitmap`, y `bmp.Size` es el tamaño del área a copiar.
  - Finalmente, la imagen capturada `bmp` se asigna a `picPantalla.Image` para su visualización.

Programa 159. Aplicación que graba sonido

Este programa permite grabar audio desde un micrófono y guardarlo en un archivo WAV.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnGrabar`, **Text:** "Grabar".
  - `Button`: **Name:** `btnDetener`, **Text:** "Detener".
  - `SaveFileDialog`: **Name:** `SaveFileDialog1`.
- **Requisito adicional:**
  - Este programa requiere la librería externa **NAudio** (generalmente se instala a través de NuGet en Visual Studio).
- **Código VB.NET:**

```
Imports NAudio.Wave
Public Class Form1
    Dim waveIn As WaveInEvent
    Dim writer As WaveFileWriter
    Private Sub btnGrabar_Click(sender As Object, e As EventArgs) Handles
btnGrabar.Click
        waveIn = New WaveInEvent()
        waveIn.WaveFormat = New WaveFormat(44100, 1) ' 44.1kHz, mono
        AddHandler waveIn.DataAvailable, Sub(s, a)
            If writer IsNot Nothing Then writer.Write(a.Buffer, 0,
a.BytesRecorded)
        End Sub
        SaveFileDialog1.Filter = "WAV|*.wav"
        If SaveFileDialog1.ShowDialog() = DialogResult.OK Then
            writer = New WaveFileWriter(SaveFileDialog1.FileName,
waveIn.WaveFormat)
            waveIn.StartRecording()
        End If
    End Sub
    Private Sub btnDetener_Click(sender As Object, e As EventArgs) Handles
btnDetener.Click
        waveIn.StopRecording()
        writer.Close()
    End Sub
End Class
```

- **Explicación:**
  - Se importa `NAudio.Wave` para usar sus clases de grabación de audio. `waveIn` y `writer` se declaran a nivel de clase para gestionar la entrada de audio y la escritura en archivo.
  - **Grabar:** Cuando se hace clic en `btnGrabar`:
    - Se inicializa `waveIn` y se configura el formato de onda (44.1 kHz, mono).
    - Se añade un manejador al evento `DataAvailable` de `waveIn`, que se dispara cuando hay datos de audio disponibles. Este manejador escribe los datos en el `writer`.
    - Se configura `SaveFileDialog1` para archivos WAV. Si el usuario selecciona una ubicación y nombre, se inicializa `writer` con el nombre del archivo y el formato de onda.
    - Finalmente, `waveIn.StartRecording()` inicia la grabación de audio.

- **Detener:** Al hacer clic en `btnDetener`, `waveIn.StopRecording()` detiene la captura de audio, y `writer.Close()` finaliza la escritura en el archivo y libera los recursos.

Programa 160. Aplicación que graba video

Este programa es un ejemplo conceptual de cómo se podría iniciar una captura de video desde una cámara web y mostrarla en un `PictureBox`. La grabación a un archivo es más compleja y requeriría una mayor implementación.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnGrabar`, **Text:** "Grabar Video".
    - `Button`: **Name:** `btnDetener`, **Text:** "Detener".
    - `PictureBox`: **Name:** `picVideo`.
- **Requisito adicional:**
    - Este programa requiere librerías externas como **AForge.Video** y **AForge.Video.DirectShow** (generalmente instaladas a través de NuGet).
- **Código VB.NET:**

```
Imports AForge.Video
Imports AForge.Video.DirectShow
Public Class Form1
    Dim videoSource As VideoCaptureDevice
    Private Sub btnGrabar_Click(sender As Object, e As EventArgs) Handles
btnGrabar.Click
        Dim videoDevices As New
FilterInfoCollection(FilterCategory.VideoInputDevice)
        videoSource = New
VideoCaptureDevice(videoDevices(0).MonikerString) ' Selecciona la primera
webcam
        AddHandler videoSource.NewFrame, Sub(s, a) picVideo.Image =
DirectCast(a.Clone(), Bitmap)
        videoSource.Start()
    End Sub
    Private Sub btnDetener_Click(sender As Object, e As EventArgs) Handles
btnDetener.Click
        If videoSource IsNot Nothing AndAlso videoSource.IsRunning Then
            videoSource.SignalToStop()
        End If
    End Sub
End Class
```

- **Explicación:**
    - Se importan `AForge.Video` y `AForge.Video.DirectShow`. `videoSource` se declara para gestionar el dispositivo de captura.
    - **Grabar Video:** Al hacer clic en `btnGrabar`:
        - Se obtiene una colección de dispositivos de entrada de video (webcams).
        - Se selecciona el primer dispositivo de video encontrado y se inicializa `videoSource` con él.
        - Se añade un manejador al evento `NewFrame` de `videoSource`, que se dispara cada vez que la cámara captura un nuevo fotograma. Este manejador convierte el fotograma a un `Bitmap` y lo asigna a `picVideo.Image` para mostrar el video en tiempo real.
        - `videoSource.Start()` inicia la captura de video.

- **Detener:** Al hacer clic en `btnDetener`, si `videoSource` está activo y funcionando, `videoSource.SignalToStop()` envía una señal para detener la captura.

Programa 161. Aplicación que reproduce listas de reproducción

Este programa permite crear una lista de reproducción de archivos de audio y reproducirlos utilizando `AxWindowsMediaPlayer`.

- **Controles necesarios en el formulario:**
  - `ListBox`: **Name:** `lstPlaylist`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Canción".
  - `Button`: **Name:** `btnReproducir`, **Text:** "Reproducir".
  - `OpenFileDialog`: **Name:** `OpenFileDialog1`.
  - `AxWindowsMediaPlayer`: **Name:** `wmp`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
        OpenFileDialog1.Filter = "MP3|*.mp3|WAV|*.wav"
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            lstPlaylist.Items.Add(OpenFileDialog1.FileName)
        End If
    End Sub
    Private Sub btnReproducir_Click(sender As Object, e As EventArgs)
Handles btnReproducir.Click
        If lstPlaylist.SelectedItem IsNot Nothing Then
            wmp.URL = lstPlaylist.SelectedItem.ToString()
            wmp.Ctlcontrols.play()
        End If
    End Sub
End Class
```

- **Explicación:**
  - Este programa utiliza el control `AxWindowsMediaPlayer`.
  - **Agregar Canción:** Cuando se hace clic en `btnAgregar`, se configura el filtro de `OpenFileDialog1` para archivos MP3 y WAV. Si el usuario selecciona un archivo, su ruta completa se añade como un elemento a `lstPlaylist`.
  - **Reproducir:** Al hacer clic en `btnReproducir`, si hay un elemento seleccionado en `lstPlaylist`, la `URL` del `AxWindowsMediaPlayer` se establece con la ruta de la canción seleccionada, y `wmp.Ctlcontrols.play()` comienza la reproducción.

Programa 162. Aplicación que descarga archivos de internet

Este programa permite al usuario descargar un archivo de una URL especificada y guardarlo en una ubicación local.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtURL`.
  - `Button`: **Name:** `btnDescargar`, **Text:** "Descargar".
  - `SaveFileDialog`: **Name:** `SaveFileDialog1`.
- **Código VB.NET:**

```
Imports System.Net
Public Class Form1
    Private Sub btnDescargar_Click(sender As Object, e As EventArgs)
Handles btnDescargar.Click
        If SaveFileDialog1.ShowDialog() = DialogResult.OK Then
            Dim client As New WebClient()
            client.DownloadFile(txtURL.Text, SaveFileDialog1.FileName)
            MessageBox.Show("Descarga completada")
        End If
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.Net` para usar la clase `WebClient`.
  - **Descargar:** Cuando se hace clic en `btnDescargar`, `SaveFileDialog1` se abre para que el usuario elija dónde guardar el archivo descargado. Si el usuario confirma, se crea una nueva instancia de `WebClient`. `client.DownloadFile()` descarga el contenido de la `URL` especificada en `txtURL.Text` y lo guarda en la ruta de archivo seleccionada por el usuario en `SaveFileDialog1.FileName`. Finalmente, un `MessageBox` notifica la finalización de la descarga.

Programa 163. Aplicación que sube archivos a servidor FTP

Este programa permite subir un archivo local a un servidor FTP, requiriendo las credenciales de usuario.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtServidor`, **Text:** "ftp://servidor.com/".
  - `TextBox`: **Name:** `txtUsuario`.
  - `TextBox`: **Name:** `txtPassword`.
  - `Button`: **Name:** `btnSubir`, **Text:** "Subir Archivo".
  - `OpenFileDialog`: **Name:** `OpenFileDialog1`.
- **Código VB.NET:**

```
Imports System.Net
Public Class Form1
    Private Sub btnSubir_Click(sender As Object, e As EventArgs) Handles
btnSubir.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            Dim request As FtpWebRequest =
CType(WebRequest.Create(txtServidor.Text &
IO.Path.GetFileName(OpenFileDialog1.FileName)), FtpWebRequest)
            request.Method = WebRequestMethods.Ftp.UploadFile
            request.Credentials = New NetworkCredential(txtUsuario.Text,
txtPassword.Text)
            Dim fileContents() As Byte =
IO.File.ReadAllBytes(OpenFileDialog1.FileName)
            request.ContentLength = fileContents.Length
            Dim requestStream = request.GetRequestStream()
            requestStream.Write(fileContents, 0, fileContents.Length)
            requestStream.Close()
            MessageBox.Show("Archivo subido correctamente")
        End If
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.Net` para operaciones de red.
  - **Subir Archivo:** Al hacer clic en `btnSubir`, `OpenFileDialog1` permite seleccionar el archivo local. Si se confirma la selección:
    - Se crea un `FtpWebRequest` construyendo la URL de destino en el servidor FTP, combinando `txtServidor.Text` con el nombre del archivo seleccionado.
    - El método de la solicitud se establece como `UploadFile`.
    - Se proporcionan las credenciales de usuario y contraseña para la autenticación FTP.
    - El contenido del archivo local se lee en una matriz de bytes.
    - Se establece la longitud del contenido, se obtiene el flujo de solicitud (`requestStream`), y los bytes del archivo se escriben en este flujo.
    - El flujo se cierra, y un `MessageBox` confirma la subida.

Programa 164. Aplicación que se conecta a una API REST

Este programa demuestra cómo realizar una simple consulta GET a una API REST y mostrar la respuesta.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtURL`.
  - `Button`: **Name:** `btnConsultar`, **Text:** "Consultar".
  - `TextBox`: **Name:** `txtResultado`, **Multiline:** `True`, **ScrollBars:** `Vertical`.
- **Código VB.NET:**

```
Imports System.Net
Public Class Form1
    Private Sub btnConsultar_Click(sender As Object, e As EventArgs)
Handles btnConsultar.Click
        Dim client As New WebClient()
        Dim response As String = client.DownloadString(txtURL.Text)
        txtResultado.Text = response
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.Net` para usar la clase `WebClient`.
  - **Consultar:** Cuando se hace clic en `btnConsultar`:
    - Se crea una nueva instancia de `WebClient`.
    - `client.DownloadString(txtURL.Text)` descarga el contenido de la URL proporcionada en `txtURL.Text` como una cadena de texto.
    - La respuesta completa de la API se muestra en `txtResultado.Text`.

Programa 165. Aplicación que se conecta a una API SOAP

Este programa es un ejemplo conceptual de cómo se interactuaría con una API SOAP, asumiendo que se ha generado un cliente de servicio.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtServicio`.
    - `Button`: **Name:** `btnConsultar`, **Text:** "Consultar".
    - `TextBox`: **Name:** `txtResultado`, **Multiline:** `True`, **ScrollBars:** `Vertical`.
- **Requisito adicional:**
    - Este programa es conceptual y requeriría agregar una referencia a un servicio web SOAP en el proyecto (usando "Agregar referencia de servicio" o "Agregar referencia web").
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnConsultar_Click(sender As Object, e As EventArgs)
Handles btnConsultar.Click
        Dim client As New ServiceReference1.MiServicioSoapClient()
        txtResultado.Text = client.MiMetodo("parametro")
    End Sub
End Class
```

- **Explicación:**
    - **Consultar:** Al hacer clic en `btnConsultar`, se crea una instancia del cliente del servicio SOAP (ej. `ServiceReference1.MiServicioSoapClient()`).
    - Luego, se llama a uno de los métodos del servicio (ej. `client.MiMetodo("parametro")`) y el resultado se muestra en `txtResultado.Text`. Este es un ejemplo genérico y los nombres exactos del cliente y los métodos dependerían del servicio SOAP específico.

Programa 166. Aplicación que consume servicios web JSON

Este programa se conecta a un servicio web, descarga datos en formato JSON y los procesa para mostrarlos.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtURL`.
  - `Button`: **Name:** `btnConsultar`, **Text:** "Consultar".
  - `TextBox`: **Name:** `txtResultado`, **Multiline:** `True`.
- **Requisito adicional:**
  - Este programa requiere la librería externa **Newtonsoft.Json** (generalmente se instala a través de NuGet).
- **Código VB.NET:**

```
Imports System.Net
Imports Newtonsoft.Json.Linq
Public Class Form1
    Private Sub btnConsultar_Click(sender As Object, e As EventArgs)
Handles btnConsultar.Click
        Dim client As New WebClient()
        Dim json As String = client.DownloadString(txtURL.Text)
        Dim obj = JObject.Parse(json)
        txtResultado.Text = obj.ToString()
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.Net` y `Newtonsoft.Json.Linq`.
  - **Consultar:** Cuando se hace clic en `btnConsultar`:
    - Se utiliza `WebClient.DownloadString()` para descargar el contenido JSON de la `txtURL.Text`.
    - `JObject.Parse(json)` analiza la cadena JSON descargada en un objeto `JObject`.
    - El objeto JSON parseado se convierte de nuevo a una cadena y se muestra en `txtResultado.Text`.

Programa 167. Aplicación que consume servicios web XML

Este programa se conecta a un servicio web, descarga datos en formato XML y los procesa para mostrarlos.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtURL`.
  - `Button`: **Name:** `btnConsultar`, **Text:** "Consultar".
  - `TextBox`: **Name:** `txtResultado`, **Multiline:** `True`.
- **Código VB.NET:**

```
Imports System.Net
Imports System.Xml
Public Class Form1
    Private Sub btnConsultar_Click(sender As Object, e As EventArgs)
Handles btnConsultar.Click
        Dim client As New WebClient()
        Dim xmlString As String = client.DownloadString(txtURL.Text)
        Dim doc As New XmlDocument()
        doc.LoadXml(xmlString)
        txtResultado.Text = doc.OuterXml
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.Net` y `System.Xml`.
  - **Consultar:** Al hacer clic en `btnConsultar`:
    - `WebClient.DownloadString()` descarga el contenido XML de la `txtURL.Text`.
    - Se crea un nuevo `XmlDocument` y se carga la cadena XML descargada en él con `doc.LoadXml()`.
    - `doc.OuterXml` devuelve la representación en cadena del documento XML completo, que se muestra en `txtResultado.Text`.

Programa 168. Cliente de chat local simple

Este programa simula un chat local básico donde los mensajes se añaden a un `ListBox`.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtMensaje`.
  - `Button`: **Name:** `btnEnviar`, **Text:** "Enviar".
  - `ListBox`: **Name:** `lstChat`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
        lstChat.Items.Add("Tú: " & txtMensaje.Text)
        txtMensaje.Clear()
    End Sub
End Class
```

- **Explicación:**
  - **Enviar:** Cuando se hace clic en `btnEnviar`, el texto de `txtMensaje` se concatena con "Tú: " y se añade como un nuevo elemento a `lstChat`. Luego, `txtMensaje` se limpia. Esto solo simula el chat localmente; no hay comunicación real con otros usuarios.

Programa 169. Cliente de chat en red local

Este programa es un ejemplo conceptual de cómo se enviarían mensajes a través de una red local utilizando `TcpClient`.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtMensaje`.
    - `Button`: **Name:** `btnEnviar`, **Text:** "Enviar".
    - `ListBox`: **Name:** `lstChat`.
- **Requisito adicional:**
    - Este programa es conceptual y requeriría una implementación de servidor de chat (no proporcionada) y una conexión activa para funcionar realmente.
- **Código VB.NET:**

```
Imports System.Net.Sockets
Imports System.Text
Imports System.Threading
Public Class Form1
    Dim client As TcpClient
    Dim stream As NetworkStream
    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
        If client IsNot Nothing Then
            Dim msg() As Byte = Encoding.ASCII.GetBytes(txtMensaje.Text)
            stream.Write(msg, 0, msg.Length)
            lstChat.Items.Add("Tú: " & txtMensaje.Text)
            txtMensaje.Clear()
        End If
    End Sub
End Class
```

- **Explicación:**
    - Se importan `System.Net.Sockets`, `System.Text` y `System.Threading`. Las variables `client` (`TcpClient`) y `stream` (`NetworkStream`) se usarían para la comunicación de red.
    - **Enviar:** Cuando se hace clic en `btnEnviar`, si el `client` está conectado, el texto de `txtMensaje` se convierte a bytes (`Encoding.ASCII.GetBytes`). Estos bytes se escriben en el `NetworkStream` (`stream.Write()`) para enviarlos al servidor. El mensaje también se añade localmente a `lstChat` y `txtMensaje` se limpia. (La configuración y conexión de `TcpClient` y `NetworkStream` no están en el código fuente proporcionado, pero serían necesarias para su funcionamiento real).

Programa 170. Cliente de chat con base de datos

Este programa almacena los mensajes de chat en una base de datos SQLite.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtMensaje`.
  - `Button`: **Name:** `btnEnviar`, **Text:** "Enviar".
  - `ListBox`: **Name:** `lstChat`.
- **Requisito adicional:**
  - Requiere la librería **System.Data.SQLite** (generalmente a través de NuGet).
- **Código VB.NET:**

```
Imports System.Data.SQLite
Public Class Form1
    Dim conn As New SQLiteConnection("Data Source=chat.db;Version=3;")
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        conn.Open()
        Dim cmd As New SQLiteCommand("CREATE TABLE IF NOT EXISTS
mensajes(usuario TEXT, mensaje TEXT)", conn)
        cmd.ExecuteNonQuery()
    End Sub
    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
        Dim cmd As New SQLiteCommand("INSERT INTO
mensajes(usuario,mensaje) VALUES('Tú',@msg)", conn)
        cmd.Parameters.AddWithValue("@msg", txtMensaje.Text)
        cmd.ExecuteNonQuery()
        lstChat.Items.Add("Tú: " & txtMensaje.Text)
        txtMensaje.Clear()
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.Data.SQLite`. `conn` es el objeto de conexión a la base de datos `chat.db`.
  - **Inicialización:** En `Form1_Load`, se abre la conexión. Se crea una tabla `mensajes` si no existe, con columnas `usuario` y `mensaje`.
  - **Enviar:** Cuando se hace clic en `btnEnviar`:
    - Se crea un `SQLiteCommand` para insertar un nuevo mensaje en la tabla `mensajes`, con "Tú" como usuario y el texto del `txtMensaje` como contenido (usando un parámetro `@msg`).
    - `cmd.ExecuteNonQuery()` ejecuta la inserción.
    - El mensaje también se añade localmente a `lstChat` y `txtMensaje` se limpia.

Programa 171. Cliente de chat con login de usuario

Este programa añade una pantalla de inicio de sesión básica para el chat, donde el usuario debe introducir un nombre y una contraseña.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtUsuario`.
    - `TextBox`: **Name:** `txtContraseña`, **PasswordChar:** `*`.
    - `Button`: **Name:** `btnLogin`, **Text:** "Iniciar Sesión".
    - `ListBox`: **Name:** `lstChat` (probablemente para el chat posterior al login).
- **Código VB.NET:**

```
Public Class Form1
    Dim usuarioLogueado As String = ""
    Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
        If txtUsuario.Text <> "" And txtContraseña.Text <> "" Then
            usuarioLogueado = txtUsuario.Text
            MessageBox.Show("Bienvenido " & usuarioLogueado)
        Else
            MessageBox.Show("Ingrese usuario y contraseña")
        End If
    End Sub
End Class
```

- **Explicación:**
    - La variable `usuarioLogueado` a nivel de clase almacena el nombre del usuario autenticado.
    - **Iniciar Sesión:** Al hacer clic en `btnLogin`, el código verifica si ambos `txtUsuario` y `txtContraseña` no están vacíos. Si lo están, `usuarioLogueado` se establece con el nombre de usuario y se muestra un mensaje de bienvenida. De lo contrario, se solicita al usuario que ingrese las credenciales. (Este login es solo una validación de campos no vacíos, no una autenticación real contra una base de datos o un usuario/contraseña predefinido como en P124).

Programa 172. Cliente de chat con historial de mensajes

Este programa carga mensajes de chat de un archivo al inicio y los guarda en el mismo archivo cada vez que se envía uno nuevo.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtMensaje`.
  - `Button`: **Name:** `btnEnviar`, **Text:** "Enviar".
  - `ListBox`: **Name:** `lstChat`.
- **Código VB.NET:**

```
Imports System.IO
Public Class Form1
    Dim historialArchivo As String = "historial.txt"
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        If File.Exists(historialArchivo) Then
            lstChat.Items.AddRange(File.ReadAllLines(historialArchivo))
        End If
    End Sub
    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
        lstChat.Items.Add("Tú: " & txtMensaje.Text)
        File.AppendAllText(historialArchivo, "Tú: " & txtMensaje.Text &
Environment.NewLine)
        txtMensaje.Clear()
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.IO`. La variable `historialArchivo` define el nombre del archivo de historial.
  - **Cargar Historial:** En `Form1_Load`, el programa verifica si el archivo `historialArchivo` existe. Si es así, lee todas las líneas del archivo y las añade a `lstChat`, mostrando el historial anterior.
  - **Enviar y Guardar:** Cuando se hace clic en `btnEnviar`, el mensaje se añade a `lstChat` y, crucialmente, se añade también al final del `historialArchivo` con `File.AppendAllText()`. `txtMensaje` se limpia.

Programa 173. Cliente de chat con emoticonos

Este programa permite al usuario insertar emoticonos predefinidos en sus mensajes de chat.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtMensaje`.
    - `Button`: **Name:** `btnEnviar`, **Text:** "Enviar".
    - `ComboBox`: **Name:** `cmbEmoticonos`, **Items:** 😀, 😳, 😎.
    - `ListBox`: **Name:** `lstChat`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
        Dim emoticono As String = cmbEmoticonos.SelectedItem
        lstChat.Items.Add("Tú: " & txtMensaje.Text & " " & emoticono)
        txtMensaje.Clear()
    End Sub
End Class
```

- **Explicación:**
    - El `ComboBox` `cmbEmoticonos` se debe poblar con los emoticonos deseados en el diseñador.
    - **Enviar:** Cuando se hace clic en `btnEnviar`, se obtiene el emoticono seleccionado de `cmbEmoticonos.SelectedItem`. El mensaje enviado a `lstChat` ahora incluye el texto del usuario más el emoticono seleccionado. `txtMensaje` se limpia.

Programa 174. Cliente de chat con archivos adjuntos

Este programa permite al usuario seleccionar un archivo local para "adjuntarlo" a un mensaje de chat, mostrando la ruta del archivo.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtMensaje`.
    - `Button`: **Name:** `btnEnviar`, **Text:** "Enviar".
    - `Button`: **Name:** `btnAdjuntar`, **Text:** "Adjuntar Archivo".
    - `OpenFileDialog`: **Name:** `OpenFileDialog1`.
    - `ListBox`: **Name:** `lstChat`.
- **Código VB.NET:**

```
Public Class Form1
    Dim archivoAdjunto As String = ""
    Private Sub btnAdjuntar_Click(sender As Object, e As EventArgs)
Handles btnAdjuntar.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            archivoAdjunto = OpenFileDialog1.FileName
            MessageBox.Show("Archivo adjuntado: " & archivoAdjunto)
        End If
    End Sub
    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
        lstChat.Items.Add("Tú: " & txtMensaje.Text & " " & archivoAdjunto)
        txtMensaje.Clear()
        archivoAdjunto = ""
    End Sub
End Class
```

- **Explicación:**
    - La variable `archivoAdjunto` a nivel de clase almacena la ruta del archivo seleccionado.
    - **Adjuntar Archivo:** Al hacer clic en `btnAdjuntar`, `OpenFileDialog1` se abre. Si se selecciona un archivo, su ruta se guarda en `archivoAdjunto` y se muestra un `MessageBox` informativo.
    - **Enviar:** Cuando se hace clic en `btnEnviar`, el mensaje que se añade a `lstChat` incluye el texto del usuario y la ruta del `archivoAdjunto`. Luego, `txtMensaje` se limpia y `archivoAdjunto` se restablece para la próxima vez. (Nota: Este ejemplo solo añade la ruta al chat, no el archivo en sí).

Programa 175. Cliente de chat con múltiples salas

Este programa permite organizar los mensajes de chat en diferentes "salas" o conversaciones, cada una con su propio historial.

- **Controles necesarios en el formulario:**
    - `ComboBox`: **Name:** `cmbSalas`, **Items:** "General", "Juegos", "Música".
    - `TextBox`: **Name:** `txtMensaje`.
    - `Button`: **Name:** `btnEnviar`, **Text:** "Enviar".
    - `ListBox`: **Name:** `lstChat`.
- **Código VB.NET:**

```
Dim salas As New Dictionary(Of String, List(Of String)) From { _
    {"General", New List(Of String)}, {"Juegos", New List(Of String)},
{"Música", New List(Of String)} _
}
Private Sub cmbSalas_SelectedIndexChanged(sender As Object, e As
EventArgs) Handles cmbSalas.SelectedIndexChanged
    lstChat.Items.Clear()
    lstChat.Items.AddRange(salas(cmbSalas.SelectedItem).ToArray())
End Sub
Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
    Dim sala As String = cmbSalas.SelectedItem
    salas(sala).Add("Tú: " & txtMensaje.Text)
    lstChat.Items.Add("Tú: " & txtMensaje.Text)
    txtMensaje.Clear()
End Sub
```

- **Explicación:**
    - Un `Dictionary` llamado `salas` se declara a nivel de clase. Cada clave del diccionario es el nombre de una sala (`String`), y el valor es una `List(Of String)` que contiene los mensajes de esa sala.
    - **Cambiar de Sala:** Cuando el usuario selecciona una sala diferente en `cmbSalas`, el evento `cmbSalas_SelectedIndexChanged` se dispara. `lstChat` se limpia y luego se llena con todos los mensajes almacenados en la lista de la sala seleccionada.
    - **Enviar Mensaje:** Al hacer clic en `btnEnviar`, el mensaje se añade a la lista correspondiente a la sala seleccionada en el diccionario `salas`. El mensaje también se añade al `lstChat` visible y `txtMensaje` se limpia.

Programa 176. Cliente de chat con videollamadas

Este programa es una implementación conceptual para indicar cómo se iniciarían las videollamadas, señalando la necesidad de librerías avanzadas.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnIniciarVideo`, **Text:** "Iniciar Videollamada".
    - `PictureBox`: **Name:** `picVideo` (para mostrar el video).
    - `AxWindowsMediaPlayer`: **Name:** `wmpVideo` (posiblemente para reproducción de video recibido).
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnIniciarVideo_Click(sender As Object, e As EventArgs)
Handles btnIniciarVideo.Click
        MessageBox.Show("Videollamada iniciada (implementación con AForge
o WebRTC necesaria)")
    End Sub
End Class
```

- **Explicación:**
    - Al hacer clic en `btnIniciarVideo`, se muestra un `MessageBox`. Este mensaje explica que una implementación real de videollamadas es compleja y requeriría el uso de librerías avanzadas como AForge o WebRTC (fuera del alcance de este ejemplo básico).

Programa 177. Cliente de chat con llamadas de voz

Similar a las videollamadas, este programa es una implementación conceptual para indicar el inicio y fin de una llamada de voz.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnIniciarVoz`, **Text:** "Iniciar Llamada".
  - `Button`: **Name:** `btnDetenerVoz`, **Text:** "Detener".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnIniciarVoz_Click(sender As Object, e As EventArgs)
Handles btnIniciarVoz.Click
        MessageBox.Show("Llamada de voz iniciada (implementación con
NAudio o VoIP necesaria)")
    End Sub
    Private Sub btnDetenerVoz_Click(sender As Object, e As EventArgs)
Handles btnDetenerVoz.Click
        MessageBox.Show("Llamada de voz detenida")
    End Sub
End Class
```

- **Explicación:**
  - Al hacer clic en `btnIniciarVoz`, se muestra un `MessageBox` conceptual, indicando que la implementación real de llamadas de voz requeriría librerías como NAudio o protocolos VoIP.
  - Al hacer clic en `btnDetenerVoz`, se muestra otro `MessageBox` conceptual, señalando la detención de la llamada.

Programa 178. Cliente de chat con notificaciones

Este programa muestra notificaciones en el área de la bandeja del sistema (system tray) para alertar al usuario sobre nuevos mensajes.

- **Controles necesarios en el formulario:**
  - `NotifyIcon`: **Name:** `NotifyIcon1`, **Icon:** `icono.ico`.
  - `Button`: **Name:** `btnNotificar`, **Text:** "Enviar Notificación".
- **Requisito adicional:**
  - Se requiere un archivo de ícono (`icono.ico`) para el `NotifyIcon`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnNotificar_Click(sender As Object, e As EventArgs)
Handles btnNotificar.Click
        NotifyIcon1.Visible = True
        NotifyIcon1.ShowBalloonTip(3000, "Chat", "Tienes un nuevo
mensaje", ToolTipIcon.Info)
    End Sub
End Class
```

- **Explicación:**
  - **Enviar Notificación:** Cuando se hace clic en `btnNotificar`, `NotifyIcon1.Visible` se establece en `True` para asegurar que el ícono sea visible en la bandeja del sistema.
  - Luego, `NotifyIcon1.ShowBalloonTip()` muestra una notificación en forma de globo. Los parámetros especifican el tiempo de duración (3000 ms), el título ("Chat"), el texto del mensaje ("Tienes un nuevo mensaje") y el tipo de ícono (Info).

Programa 179. Cliente de chat con moderadores

Este programa introduce una funcionalidad básica de moderación para bloquear usuarios en el chat.

- **Controles necesarios en el formulario:**
    - `ListBox`: **Name:** `lstChat`.
    - `Button`: **Name:** `btnBloquear`, **Text:** "Bloquear Usuario".
    - `TextBox`: **Name:** `txtUsuario`.
- **Código VB.NET:**

```
Dim usuariosBloqueados As New List(Of String)
Private Sub btnBloquear_Click(sender As Object, e As EventArgs) Handles
btnBloquear.Click
    usuariosBloqueados.Add(txtUsuario.Text)
    MessageBox.Show("Usuario " & txtUsuario.Text & " bloqueado")
End Sub
```

- **Explicación:**
    - Se declara una lista `usuariosBloqueados` a nivel de clase para almacenar los nombres de usuario que han sido bloqueados.
    - **Bloquear Usuario:** Cuando se hace clic en `btnBloquear`, el nombre de usuario de `txtUsuario.Text` se añade a la lista `usuariosBloqueados`. Un `MessageBox` confirma que el usuario ha sido bloqueado. (Esta implementación es conceptual; la lógica para impedir que un usuario bloqueado envíe/reciba mensajes no está en el código proporcionado).

Programa 180. Cliente de chat con administración

Este programa implementa funciones básicas para agregar y eliminar usuarios en una lista de administración de chat.

- **Controles necesarios en el formulario:**
    - `ListBox`: **Name:** `lstUsuarios`.
    - `Button`: **Name:** `btnEliminar`, **Text:** "Eliminar Usuario".
    - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Usuario".
    - `TextBox`: **Name:** `txtUsuario`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstUsuarios.Items.Add(txtUsuario.Text)
End Sub
Private Sub btnEliminar_Click(sender As Object, e As EventArgs) Handles
btnEliminar.Click
    If lstUsuarios.SelectedItem IsNot Nothing Then
lstUsuarios.Items.Remove(lstUsuarios.SelectedItem)
End Sub
```

- **Explicación:**
    - **Agregar Usuario:** Al hacer clic en `btnAgregar`, el texto del `txtUsuario` se añade como un nuevo elemento a `lstUsuarios`.
    - **Eliminar Usuario:** Al hacer clic en `btnEliminar`, se verifica si hay un elemento seleccionado en `lstUsuarios`. Si lo hay, `lstUsuarios.Items.Remove()` elimina el elemento seleccionado de la lista. (Esta funcionalidad es puramente visual; no interactúa con un sistema de usuarios persistente).

Programa 181. Cliente de chat con cifrado de mensajes

Este programa implementa el cifrado básico de mensajes en un cliente de chat utilizando la codificación Base64.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtMensaje`.
  - `Button`: **Name:** `btnEnviar`, **Text:** "Enviar".
  - `ListBox`: **Name:** `lstChat`.
- **Código VB.NET:**

```
Imports System.Text
Public Class Form1
    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
        Dim mensaje As String = txtMensaje.Text
        Dim mensajeCifrado As String =
Convert.ToBase64String(Encoding.UTF8.GetBytes(mensaje))
        lstChat.Items.Add("Tú: " & mensajeCifrado)
        txtMensaje.Clear()
    End Sub
End Class
```

- **Explicación:**
  - Se importa el espacio de nombres `System.Text` para las operaciones de codificación.
  - Cuando el usuario hace clic en `btnEnviar`, el texto del `txtMensaje` se guarda en la variable `mensaje`.
  - `Encoding.UTF8.GetBytes(mensaje)` convierte el texto a una matriz de bytes utilizando la codificación UTF-8.
  - `Convert.ToBase64String()` toma esta matriz de bytes y la convierte en una **cadena Base64**, que es una representación cifrada del mensaje original.
  - El mensaje cifrado se añade al `lstChat` precedido por "Tú: ".
  - Finalmente, el `txtMensaje` se vacía.

Programa 182. Cliente de chat con recuperación de historial

Este programa carga y guarda automáticamente el historial de mensajes de chat en un archivo de texto, y permite recuperarlo manualmente.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtMensaje`.
    - `Button`: **Name:** `btnEnviar`, **Text:** "Enviar".
    - `Button`: **Name:** `btnRecuperar`, **Text:** "Recuperar Historial".
    - `ListBox`: **Name:** `lstChat`.
- **Código VB.NET:**

```
Imports System.IO
Public Class Form1
    Dim archivoHistorial As String = "historial_chat.txt"
    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
        lstChat.Items.Add("Tú: " & txtMensaje.Text)
        File.AppendAllText(archivoHistorial, "Tú: " & txtMensaje.Text &
Environment.NewLine)
        txtMensaje.Clear()
    End Sub
    Private Sub btnRecuperar_Click(sender As Object, e As EventArgs)
Handles btnRecuperar.Click
        If File.Exists(archivoHistorial) Then
            lstChat.Items.Clear()
            lstChat.Items.AddRange(File.ReadAllLines(archivoHistorial))
        End If
    End Sub
End Class
```

- **Explicación:**
    - Se importa `System.IO` para las operaciones de archivo.
    - La variable `archivoHistorial` define la ruta del archivo donde se guardará el historial.
    - **Enviar Mensaje:** Al hacer clic en `btnEnviar`, el mensaje se añade al `lstChat` y se **guarda en el archivo de historial** utilizando `File.AppendAllText()`, que añade el texto al final del archivo. El `txtMensaje` se limpia.
    - **Recuperar Historial:** Al hacer clic en `btnRecuperar`, se verifica si el `archivoHistorial` existe. Si existe, `lstChat` se limpia, y `File.ReadAllLines()` lee todas las líneas del archivo, las cuales se añaden al `lstChat` para mostrar el historial recuperado.

Programa 183. Cliente de chat con estadísticas

Este programa calcula y muestra el número total de mensajes en el chat como una estadística básica.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnEstadisticas`, **Text:** "Mostrar Estadísticas".
  - `ListBox`: **Name:** `lstChat`.
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnEstadisticas_Click(sender As Object, e As EventArgs)
Handles btnEstadisticas.Click
        Dim totalMensajes As Integer = lstChat.Items.Count
        MessageBox.Show("Total de mensajes: " & totalMensajes)
    End Sub
End Class
```

- **Explicación:**
  - Cuando el usuario hace clic en `btnEstadisticas`, el evento `btnEstadisticas_Click` se activa.
  - `lstChat.Items.Count` obtiene el número de elementos (mensajes) actualmente en el `ListBox`.
  - Un `MessageBox` muestra este `totalMensajes` al usuario.

Programa 184. Cliente de chat con exportación de mensajes

Este programa permite al usuario exportar el contenido del `ListBox` del chat a un archivo de texto en una ubicación deseada.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnExportar`, **Text:** "Exportar".
  - `SaveFileDialog`: **Name:** `SaveFileDialog1`.
  - `ListBox`: **Name:** `lstChat`.
- **Código VB.NET:**

```
Imports System.IO
Public Class Form1
    Private Sub btnExportar_Click(sender As Object, e As EventArgs)
Handles btnExportar.Click
        If SaveFileDialog1.ShowDialog() = DialogResult.OK Then
            File.WriteAllLines(SaveFileDialog1.FileName,
lstChat.Items.Cast(Of String)())
            MessageBox.Show("Mensajes exportados")
        End If
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.IO` para la manipulación de archivos.
  - Al hacer clic en `btnExportar`, `SaveFileDialog1.ShowDialog()` abre un cuadro de diálogo para que el usuario elija la ubicación y el nombre del archivo de exportación.
  - Si el usuario confirma (`DialogResult.OK`), `File.WriteAllLines()` escribe todos los elementos del `lstChat` (convertidos a `String` con `Cast(Of String)()`) como líneas separadas en el archivo especificado por `SaveFileDialog1.FileName`.
  - Se muestra un `MessageBox` para confirmar la exportación.

Programa 185. Cliente de chat con importación de mensajes

Este programa permite al usuario importar mensajes de chat desde un archivo de texto y cargarlos en el `ListBox` del chat.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnImportar`, **Text:** "Importar".
  - `OpenFileDialog`: **Name:** `OpenFileDialog1`.
  - `ListBox`: **Name:** `lstChat`.
- **Código VB.NET:**

```
Imports System.IO
Public Class Form1
    Private Sub btnImportar_Click(sender As Object, e As EventArgs)
Handles btnImportar.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            lstChat.Items.Clear()

lstChat.Items.AddRange(File.ReadAllLines(OpenFileDialog1.FileName))
            MessageBox.Show("Mensajes importados")
        End If
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.IO` para operaciones de archivo.
  - Cuando el usuario hace clic en `btnImportar`, `OpenFileDialog1.ShowDialog()` abre un cuadro de diálogo para que el usuario seleccione un archivo de texto.
  - Si se selecciona un archivo (`DialogResult.OK`), `lstChat` se limpia y `File.ReadAllLines()` lee todas las líneas del archivo. Estas líneas se añaden al `lstChat` utilizando `lstChat.Items.AddRange()`.
  - Un `MessageBox` confirma la importación de los mensajes.

Programa 186. Cliente de correo electrónico básico

Este programa es una implementación conceptual para preparar un correo electrónico, pero el envío real requiere la configuración de SMTP.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtPara`.
    - `TextBox`: **Name:** `txtAsunto`.
    - `TextBox`: **Name:** `txtCuerpo`, **Multiline:** `True`.
    - `Button`: **Name:** `btnEnviar`, **Text:** "Enviar".
- **Código VB.NET:**

```
Imports System.Net.Mail
Public Class Form1
    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
        Dim mail As New MailMessage()
        mail.To.Add(txtPara.Text)
        mail.Subject = txtAsunto.Text
        mail.Body = txtCuerpo.Text
        MessageBox.Show("Correo preparado (implementación real requiere
SMTP)")
    End Sub
End Class
```

- **Explicación:**
    - Se importa `System.Net.Mail` para trabajar con correos electrónicos.
    - Cuando se hace clic en `btnEnviar`, se crea una nueva instancia de `MailMessage`.
    - Las propiedades `To`, `Subject` y `Body` del objeto `mail` se establecen con el texto de los respectivos `TextBox`s.
    - Finalmente, un `MessageBox` indica que el correo ha sido "preparado" y aclara que la implementación real del envío requiere una configuración SMTP.

Programa 187. Cliente de correo con envío SMTP

Este programa implementa el envío de correos electrónicos utilizando el protocolo SMTP (Simple Mail Transfer Protocol) con autenticación.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtPara`.
  - `TextBox`: **Name:** `txtAsunto`.
  - `TextBox`: **Name:** `txtCuerpo`, **Multiline:** `True`.
  - `TextBox`: **Name:** `txtSMTP` (servidor SMTP).
  - `TextBox`: **Name:** `txtUsuario` (correo remitente).
  - `TextBox`: **Name:** `txtPassword`, **PasswordChar:** *.
  - `Button`: **Name:** `btnEnviar`, **Text:** "Enviar".
- **Código VB.NET:**

```
Imports System.Net.Mail
Public Class Form1
    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
        Dim smtp As New SmtpClient(txtSMTP.Text)
        smtp.Credentials = New Net.NetworkCredential(txtUsuario.Text,
txtPassword.Text)
        Dim mail As New MailMessage(txtUsuario.Text, txtPara.Text,
txtAsunto.Text, txtCuerpo.Text)
        smtp.Send(mail)
        MessageBox.Show("Correo enviado")
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.Net.Mail`.
  - Cuando se hace clic en `btnEnviar`:
    - Se crea una instancia de `SmtpClient`, proporcionando el servidor SMTP desde `txtSMTP.Text`.
    - Se establecen las credenciales (`NetworkCredential`) para la autenticación en el servidor SMTP, utilizando el usuario (`txtUsuario.Text`) y la contraseña (`txtPassword.Text`).
    - Se crea un `MailMessage` especificando el remitente, el destinatario, el asunto y el cuerpo del correo.
    - `smtp.Send(mail)` envía el correo electrónico a través del servidor SMTP configurado.
    - Un `MessageBox` confirma que el correo ha sido enviado.

Programa 188. Cliente de correo con recepción POP3

Este programa es un ejemplo conceptual de cómo se recibirían correos electrónicos utilizando el protocolo POP3 (Post Office Protocol 3), haciendo uso de una librería externa como OpenPop.NET.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtServidor`.
    - `TextBox`: **Name:** `txtUsuario`.
    - `TextBox`: **Name:** `txtPassword`, **PasswordChar:** `*`.
    - `Button`: **Name:** `btnRecibir`, **Text:** "Recibir".
    - `ListBox`: **Name:** `lstCorreos`.
- **Requisito adicional:**
    - Este programa es conceptual y requeriría la instalación de la librería **OpenPop.NET** (generalmente a través de NuGet) para su funcionamiento real.
- **Código VB.NET:**

```
Imports OpenPop.Pop3
Public Class Form1
    Private Sub btnRecibir_Click(sender As Object, e As EventArgs) Handles
btnRecibir.Click
        Dim client As New Pop3Client()
        client.Connect(txtServidor.Text, 110, False)
        client.Authenticate(txtUsuario.Text, txtPassword.Text)
        Dim count = client.GetMessageCount()
        lstCorreos.Items.Clear()
        For i As Integer = 1 To count
            Dim msg = client.GetMessage(i)
            lstCorreos.Items.Add(msg.Headers.Subject)
        Next
        ' (Opcional: client.Disconnect() para cerrar la conexión)
    End Sub
End Class
```

- **Explicación:**
    - Se importa `OpenPop.Pop3`.
    - **Recibir Correos:** Al hacer clic en `btnRecibir`:
        - Se crea una instancia de `Pop3Client`.
        - El cliente se conecta al servidor POP3 (`txtServidor.Text`) en el puerto 110 sin SSL (`False`).
        - Se autentica con el `txtUsuario.Text` y `txtPassword.Text`.
        - `client.GetMessageCount()` obtiene el número total de mensajes en el buzón.
        - `lstCorreos` se limpia, y un bucle `For` itera desde 1 hasta el número de mensajes.
        - En cada iteración, `client.GetMessage(i)` descarga el mensaje `i`, y su asunto (`msg.Headers.Subject`) se añade a `lstCorreos`.

Programa 189. Cliente de correo con conexión IMAP

Este programa es un ejemplo conceptual de cómo se interactuaría con un servidor de correo IMAP (Internet Message Access Protocol) utilizando una librería externa como MailKit.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtServidor`.
    - `TextBox`: **Name:** `txtUsuario`.
    - `TextBox`: **Name:** `txtPassword`, **PasswordChar:** `*`.
    - `Button`: **Name:** `btnRecibir`, **Text:** "Recibir".
    - `ListBox`: **Name:** `lstCorreos`.
- **Requisito adicional:**
    - Este programa es conceptual y requeriría la instalación de la librería **MailKit** (generalmente a través de NuGet) para su funcionamiento real.
- **Código VB.NET:**

```
Imports MailKit.Net.Imap
Imports MailKit
Imports MimeKit
Public Class Form1
    Private Sub btnRecibir_Click(sender As Object, e As EventArgs) Handles
btnRecibir.Click
        Using client As New ImapClient()
            client.Connect(txtServidor.Text, 993, True)
            client.Authenticate(txtUsuario.Text, txtPassword.Text)
            Dim inbox = client.Inbox
            inbox.Open(FolderAccess.ReadOnly)
            lstCorreos.Items.Clear()
            For i = 0 To inbox.Count - 1
                lstCorreos.Items.Add(inbox.GetMessage(i).Subject)
            Next
            client.Disconnect(True)
        End Using
    End Sub
End Class
```

- **Explicación:**
    - Se importan `MailKit.Net.Imap`, `MailKit` y `MimeKit`.
    - **Recibir Correos:** Al hacer clic en `btnRecibir`:
        - Se utiliza un bloque `Using` para asegurar la correcta disposición de `ImapClient`.
        - El cliente se conecta al servidor IMAP (`txtServidor.Text`) en el puerto 993 con SSL (`True`).
        - Se autentica con el usuario y la contraseña.
        - Se accede a la bandeja de entrada (`inbox`) y se abre en modo de solo lectura (`FolderAccess.ReadOnly`).
        - `lstCorreos` se limpia, y un bucle `For` itera sobre los mensajes de la bandeja de entrada.
        - Para cada mensaje, se obtiene su asunto (`inbox.GetMessage(i).Subject`) y se añade a `lstCorreos`.
        - Finalmente, el cliente se desconecta (`client.Disconnect(True)`).

Programa 190. Cliente de correo con adjuntos

Este programa permite al usuario adjuntar uno o más archivos a un correo electrónico antes de enviarlo.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtPara`.
  - `TextBox`: **Name:** `txtAsunto`.
  - `TextBox`: **Name:** `txtCuerpo`, **Multiline:** `True`.
  - `Button`: **Name:** `btnAdjuntar`, **Text:** "Adjuntar Archivo".
  - `Button`: **Name:** `btnEnviar`, **Text:** "Enviar".
  - `OpenFileDialog`: **Name:** `OpenFileDialog1`.
  - `ListBox`: **Name:** `lstAdjuntos` (para mostrar los archivos adjuntos).
- **Código VB.NET:**

```
Imports System.Net.Mail
Public Class Form1
    Dim archivosAdjuntos As New List(Of String)
    Private Sub btnAdjuntar_Click(sender As Object, e As EventArgs)
Handles btnAdjuntar.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            archivosAdjuntos.Add(OpenFileDialog1.FileName)

lstAdjuntos.Items.Add(IO.Path.GetFileName(OpenFileDialog1.FileName))
        End If
    End Sub
    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
        Dim mail As New MailMessage()
        mail.To.Add(txtPara.Text)
        mail.Subject = txtAsunto.Text
        mail.Body = txtCuerpo.Text
        For Each file In archivosAdjuntos
            mail.Attachments.Add(New Attachment(file))
        Next
        MessageBox.Show("Correo preparado con adjuntos (SMTP necesario
para envío real)")
    End Sub
End Class
```

- **Explicación:**
  - Se importa `System.Net.Mail`. La lista `archivosAdjuntos` (a nivel de clase) guarda las rutas de los archivos seleccionados.
  - **Adjuntar Archivo:** Al hacer clic en `btnAdjuntar`, `OpenFileDialog1` se abre. Si el usuario selecciona un archivo, su ruta completa se añade a la lista `archivosAdjuntos`, y solo su nombre de archivo se añade a `lstAdjuntos` para visualización.
  - **Enviar:** Cuando se hace clic en `btnEnviar`, se crea un `MailMessage` con el destinatario, asunto y cuerpo. Luego, un bucle `For Each` itera sobre `archivosAdjuntos`, creando un nuevo objeto `Attachment` para cada archivo y añadiéndolo a la colección `mail.Attachments`. Un `MessageBox` confirma que el correo está "preparado con adjuntos" y recuerda la necesidad de una configuración SMTP para el envío real.

Programa 200. Cliente de correo con múltiples cuentas

Este programa conceptual simula el envío de correos desde diferentes cuentas seleccionadas por el usuario.

- **Controles necesarios en el formulario:**
  - `ComboBox`: **Name:** `cmbCuentas`, **Items:** "Cuenta1", "Cuenta2".
  - `TextBox`: **Name:** `txtPara`.
  - `TextBox`: **Name:** `txtAsunto`.
  - `TextBox`: **Name:** `txtCuerpo`, **Multiline:** `True`.
  - `Button`: **Name:** `btnEnviar`, **Text:** "Enviar".
- **Código VB.NET:**

```
Public Class Form1
    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
        Dim cuentaSeleccionada As String =
cmbCuentas.SelectedItem.ToString()
        MessageBox.Show("Correo enviado desde " & cuentaSeleccionada & " a
" & txtPara.Text)
    End Sub
End Class
```

- **Explicación:**
  - Cuando se hace clic en `btnEnviar`, el programa obtiene la `cuentaSeleccionada` del `cmbCuentas`.
  - Un `MessageBox` muestra un mensaje conceptual indicando que el correo sería enviado desde la cuenta seleccionada al destinatario, sin realizar un envío real. Para una implementación real, se necesitarían credenciales SMTP para cada cuenta.

Programa 201. Cliente de correo con filtros de mensajes

Este programa conceptual simula la aplicación de un filtro de texto a una lista de correos, mostrando los que coinciden.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtFiltro`.
  - `Button`: **Name:** `btnAplicarFiltro`, **Text:** "Aplicar Filtro".
  - `ListBox`: **Name:** `lstCorreos` (asumir que contiene elementos para filtrar).
- **Código VB.NET:**

```
Private Sub btnAplicarFiltro_Click(sender As Object, e As EventArgs)
Handles btnAplicarFiltro.Click
    Dim filtro As String = txtFiltro.Text.ToLower()
    For Each item As String In lstCorreos.Items
        If item.ToLower().Contains(filtro) Then
            MessageBox.Show("Correo filtrado: " & item)
        End If
    Next
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnAplicarFiltro`, el texto del `txtFiltro` se convierte a minúsculas para realizar una búsqueda sin distinción de mayúsculas y minúsculas.
  - Un bucle `For Each` itera sobre cada `item` en `lstCorreos.Items`.
  - Si el `item` (convertido a minúsculas) contiene el `filtro`, se muestra un `MessageBox` con el texto del correo filtrado.

Programa 202. Cliente de correo con carpetas personalizadas

Este programa permite al usuario crear nuevas carpetas personalizadas que se muestran en un `TreeView`.

- **Controles necesarios en el formulario:**
  - `TreeView`: **Name:** `treeCarpetas`.
  - `ListBox`: **Name:** `lstCorreos` (para mostrar correos dentro de las carpetas).
  - `Button`: **Name:** `btnCrearCarpeta`, **Text:** "Crear Carpeta".
  - `TextBox`: **Name:** `txtNuevaCarpeta`.
- **Código VB.NET:**

```
Private Sub btnCrearCarpeta_Click(sender As Object, e As EventArgs)
Handles btnCrearCarpeta.Click
    treeCarpetas.Nodes.Add(txtNuevaCarpeta.Text)
    txtNuevaCarpeta.Clear()
End Sub
```

- **Explicación:**
  - Cuando el usuario hace clic en `btnCrearCarpeta`, el texto del `txtNuevaCarpeta` se añade como un nuevo nodo (carpeta) a `treeCarpetas.Nodes`.
  - El `txtNuevaCarpeta` se limpia después de agregar la carpeta.

Programa 203. Cliente de correo con notificaciones

Este programa muestra una notificación en el área de la bandeja del sistema (system tray) para alertar al usuario sobre nuevos mensajes de correo.

- **Controles necesarios en el formulario:**
  - `NotifyIcon`: **Name:** `NotifyIcon1`, **Icon:** `icono.ico`.
  - `Button`: **Name:** `btnNotificar`, **Text:** "Notificar".
- **Código VB.NET:**

```
Private Sub btnNotificar_Click(sender As Object, e As EventArgs) Handles
btnNotificar.Click
    NotifyIcon1.Visible = True
    NotifyIcon1.ShowBalloonTip(3000, "Correo", "Tienes un nuevo mensaje",
ToolTipIcon.Info)
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnNotificar`, la propiedad `NotifyIcon1.Visible` se establece en `True` para asegurar que el ícono sea visible en la bandeja del sistema.
  - `NotifyIcon1.ShowBalloonTip()` muestra una notificación en forma de globo con una duración de 3 segundos (3000 ms), un título de "Correo", el mensaje "Tienes un nuevo mensaje" y un ícono de información.

Programa 204. Cliente de correo con contactos

Este programa permite al usuario agregar contactos (nombre y correo electrónico) a una lista.

- **Controles necesarios en el formulario:**
  - `ListBox`: **Name:** `lstContactos`.
  - `TextBox`: **Name:** `txtNombre`.
  - `TextBox`: **Name:** `txtCorreo`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Contacto".
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstContactos.Items.Add(txtNombre.Text & " <" & txtCorreo.Text & ">")
    txtNombre.Clear()
    txtCorreo.Clear()
End Sub
```

- **Explicación:**
  - Cuando el usuario hace clic en `btnAgregar`, el nombre y el correo electrónico se concatenan en un formato "Nombre correo@dominio.com" y se añaden como un nuevo elemento a `lstContactos`.
  - Los `TextBox` `txtNombre` y `txtCorreo` se limpian después de añadir el contacto.

Programa 205. Cliente de correo con calendario

Este programa integra un control de calendario (`MonthCalendar`) para que el usuario pueda seleccionar una fecha y mostrarla.

- **Controles necesarios en el formulario:**
  - `MonthCalendar`: **Name:** `calendar`.
  - `Button`: **Name:** `btnSeleccionar`, **Text:** "Seleccionar Fecha".
  - `TextBox`: **Name:** `txtFechaSeleccionada`.
- **Código VB.NET:**

```
Private Sub btnSeleccionar_Click(sender As Object, e As EventArgs) Handles
btnSeleccionar.Click
    txtFechaSeleccionada.Text =
calendar.SelectionStart.ToShortDateString()
End Sub
```

- **Explicación:**
  - Cuando el usuario hace clic en `btnSeleccionar`, la propiedad `calendar.SelectionStart` (que representa la fecha seleccionada en el `MonthCalendar`) se convierte a una cadena de fecha corta (`ToShortDateString()`) y se muestra en `txtFechaSeleccionada.Text`.

Programa 206. Cliente de correo con recordatorios

Este programa permite al usuario agregar recordatorios con una fecha y un mensaje, mostrándolos en un `ListBox`.

- **Controles necesarios en el formulario:**
    - `DateTimePicker`: **Name:** `dtpRecordatorio`.
    - `TextBox`: **Name:** `txtMensaje`.
    - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Recordatorio".
    - `ListBox`: **Name:** `lstRecordatorios`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstRecord limpia después de agregar el recordatorio.
```

Programa 207. Cliente de correo con sincronización

Este programa es un ejemplo conceptual de cómo se iniciaría una sincronización de correos con un servidor.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnSincronizar`, **Text:** "Sincronizar".
    - `ListBox`: **Name:** `lstCorreos` (asumir que mostraría los correos sincronizados).
- **Código VB.NET:**

```
Private Sub btnSincronizar_Click(sender As Object, e As EventArgs) Handles
btnSincronizar.Click
    MessageBox.Show("Sincronización de correos iniciada (implementación
real requiere servidor)")
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnSincronizar`, se muestra un `MessageBox` conceptual, indicando que la sincronización se ha iniciado y que una implementación real requeriría la interacción con un servidor de correo.

Programa 208. Cliente de correo con cifrado

Este programa permite al usuario cifrar el cuerpo de un mensaje de correo utilizando una codificación Base64 simple.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtCuerpo`, **Multiline:** `True`.
  - `Button`: **Name:** `btnCifrar`, **Text:** "Cifrar".
  - `TextBox`: **Name:** `txtCifrado`, **Multiline:** `True`.
- **Código VB.NET:**

```
Imports System.Text
Private Sub btnCifrar_Click(sender As Object, e As EventArgs) Handles
btnCifrar.Click
    txtCifrado.Text =
Convert.ToBase64String(Encoding.UTF8.GetBytes(txtCuerpo.Text))
End Sub
```

- **Explicación:**
  - Se importa `System.Text` para las operaciones de codificación.
  - Cuando el usuario hace clic en `btnCifrar`, el texto del `txtCuerpo` se convierte a una matriz de bytes utilizando UTF-8, y luego se codifica en Base64 con `Convert.ToBase64String()`.
  - El texto cifrado resultante se muestra en `txtCifrado.Text`.

Programa 209. Cliente de correo con firmas personalizadas

Este programa permite al usuario añadir una firma predefinida al final del cuerpo de su mensaje de correo.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtCuerpo`, **Multiline:** `True`.
  - `TextBox`: **Name:** `txtFirma`.
  - `Button`: **Name:** `btnEnviar`, **Text:** "Enviar".
- **Código VB.NET:**

```
Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
    Dim mensajeFinal As String = txtCuerpo.Text & Environment.NewLine &
txtFirma.Text
    MessageBox.Show("Correo listo para enviar: " & Environment.NewLine &
mensajeFinal)
End Sub
```

- **Explicación:**
  - Cuando el usuario hace clic en `btnEnviar`, el texto del `txtCuerpo` se concatena con un salto de línea (`Environment.NewLine`) y el texto del `txtFirma` para crear el `mensajeFinal`.
  - Un `MessageBox` muestra este `mensajeFinal` completo, simulando que el correo está listo para ser enviado.

Programa 210. Sistema de login con logs de actividad

Este programa implementa un sistema de login que registra cada intento de inicio de sesión en un archivo de log de actividad.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtUsuario`.
    - `TextBox`: **Name:** `txtContraseña`, **PasswordChar:** `*`.
    - `Button`: **Name:** `btnLogin`, **Text:** "Iniciar Sesión".
- **Código VB.NET:**

```
Imports System.IO
Public Class Form1
    Dim logArchivo As String = "log_actividad.txt"
    Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
        If txtUsuario.Text <> "" And txtContraseña.Text <> "" Then
            File.AppendAllText(logArchivo, DateTime.Now.ToString() & " - "
& txtUsuario.Text & " inició sesión" & Environment.NewLine)
            MessageBox.Show("Bienvenido " & txtUsuario.Text)
        Else
            MessageBox.Show("Ingrese usuario y contraseña")
        End If
    End Sub
End Class
```

- **Explicación:**
    - Se importa `System.IO` para la gestión de archivos.
    - La variable `logArchivo` define el nombre del archivo de log.
    - Cuando el usuario hace clic en `btnLogin`:
        - Se verifica que los campos `txtUsuario` y `txtContraseña` no estén vacíos.
        - Si no están vacíos, se registra la fecha y hora actuales (`DateTime.Now.ToString()`), el nombre de usuario y el evento "inició sesión" en el `logArchivo` utilizando `File.AppendAllText()`.
        - Luego, se muestra un `MessageBox` de bienvenida.
        - Si los campos están vacíos, se solicita al usuario que los ingrese. (Este login es conceptual y no realiza una autenticación real contra credenciales predefinidas o una base de datos).

Programa 211. Sistema de login con auditoría

Este programa registra cada intento de inicio de sesión en un `ListBox` para fines de auditoría, indicando la fecha, hora y el usuario que intentó acceder.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtUsuario`.
    - `TextBox`: **Name:** `txtContraseña`, **PasswordChar:** `*`.
    - `Button`: **Name:** `btnLogin`, **Text:** "Iniciar Sesión".
    - `ListBox`: **Name:** `lstAuditoria`.
- **Código VB.NET:**

```
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    Dim auditoria As String = DateTime.Now.ToString() & " - Usuario: " &
txtUsuario.Text & " intentó iniciar sesión"
    lstAuditoria.Items.Add(auditoria)
    MessageBox.Show("Intento registrado")
End Sub
```

- **Explicación:**
    - Cuando se hace clic en `btnLogin`, el evento `btnLogin_Click` se activa.
    - Se construye una cadena `auditoria` que incluye la fecha y hora actuales (`DateTime.Now.ToString()`), el nombre de usuario de `txtUsuario.Text`, y la acción "intentó iniciar sesión".
    - Esta cadena se añade como un nuevo elemento al `lstAuditoria`.
    - Finalmente, un `MessageBox` notifica al usuario que el intento ha sido registrado.

      Este programa se centra en el registro del intento, no en la validación de credenciales reales.

Programa 212. Sistema de login con autenticación externa

Este programa simula un proceso de autenticación contra un servicio o sistema externo. Aunque el código solo incluye una validación local, el concepto es que las credenciales se verificarían en otro lugar.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtUsuario`.
    - `TextBox`: **Name:** `txtContraseña`, **PasswordChar:** `*`.
    - `Button`: **Name:** `btnLogin`, **Text:** "Iniciar Sesión".
- **Código VB.NET:**

```
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    ' Autenticación externa simulada
    If txtUsuario.Text = "admin" And txtContraseña.Text = "1234" Then
        MessageBox.Show("Autenticación exitosa (externa)")
    Else
        MessageBox.Show("Autenticación fallida")
    End If
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnLogin`, el evento `btnLogin_Click` se ejecuta.
    - El código simula la verificación de un usuario "admin" con la contraseña "1234".
    - Si las credenciales coinciden, se muestra un `MessageBox` indicando "Autenticación exitosa (externa)"; de lo contrario, muestra "Autenticación fallida". En una aplicación real, esta verificación implicaría una llamada a una API de autenticación, un servidor LDAP, o un proveedor de identidad.

Programa 213. Sistema de login con validación de IP

Este programa implementa una capa de seguridad adicional al verificar la dirección IP del usuario que intenta iniciar sesión, permitiendo el acceso solo desde IPs autorizadas.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtUsuario`.
    - `TextBox`: **Name:** `txtContraseña`, **PasswordChar:** `*`.
    - `Button`: **Name:** `btnLogin`, **Text:** "Iniciar Sesión".
- **Código VB.NET:**

```
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    Dim ipActual As String = "192.168.1.100" ' Simulada
    Dim ipPermitida As String = "192.168.1.100"
    If ipActual = ipPermitida Then
        MessageBox.Show("IP válida, acceso permitido")
    Else
        MessageBox.Show("IP no permitida")
    End If
End Sub
```

- **Explicación:**
    - Cuando se hace clic en `btnLogin`, el evento `btnLogin_Click` se activa.
    - El programa define una `ipActual` simulada y una `ipPermitida`.
    - Se compara `ipActual` con `ipPermitida`. Si coinciden, se muestra un `MessageBox` de "IP válida, acceso permitido"; de lo contrario, "IP no permitida". En una aplicación real, `ipActual` se obtendría de la configuración de red o de una solicitud HTTP, y `ipPermitida` provendría de una configuración de seguridad o base de datos.

Programa 214. Sistema de login con tiempo de expiración

Este programa simula un tiempo de expiración de sesión, donde el acceso se concede solo si el intento de login ocurre antes de una fecha y hora límite predefinidas.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtUsuario`.
    - `TextBox`: **Name:** `txtContraseña`, **PasswordChar:** `*`.
    - `Button`: **Name:** `btnLogin`, **Text:** "Iniciar Sesión".
- **Código VB.NET:**

```
Dim tiempoExpiracion As DateTime = DateTime.Now.AddMinutes(30)
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    If DateTime.Now <= tiempoExpiracion Then
        MessageBox.Show("Sesión iniciada correctamente")
    Else
        MessageBox.Show("Tiempo de sesión expirado")
    End If
End Sub
```

- **Explicación:**
    - La variable `tiempoExpiracion` se declara a nivel de clase y se inicializa con la hora actual más 30 minutos.
    - Cuando se hace clic en `btnLogin`, el código compara la `DateTime.Now` (hora actual) con `tiempoExpiracion`.
    - Si la hora actual es anterior o igual al `tiempoExpiracion`, se muestra "Sesión iniciada correctamente". Si el tiempo ha pasado, se muestra "Tiempo de sesión expirado". Este es un ejemplo simplificado de una sesión de corta duración.

Programa 215. Sistema de login con recordatorio de sesión

Este programa utiliza un `Timer` para mostrar un recordatorio periódico al usuario después de iniciar sesión, instándolo a cerrar la sesión si ha terminado.

- **Controles necesarios en el formulario:**
    - `Timer`: **Name:** `Timer1`, **Interval:** `60000` (1 min).
    - `Label`: **Name:** `lblRecordatorio`.
    - `TextBox`: **Name:** `txtUsuario`.
    - `TextBox`: **Name:** `txtContraseña`, **PasswordChar:** `*`.
    - `Button`: **Name:** `btnLogin`, **Text:** "Iniciar Sesión".
- **Código VB.NET:**

```
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
    lblRecordatorio.Text = "Recuerda cerrar sesión si terminas"
End Sub
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    Timer1.Start()
    MessageBox.Show("Sesión iniciada")
End Sub
```

- **Explicación:**
    - El `Timer1` se configura con un intervalo de 60000 milisegundos (1 minuto).
    - Cuando el usuario hace clic en `btnLogin`, el `Timer1` se inicia.
    - Cada vez que el `Timer1_Tick` se dispara (cada minuto), el `lblRecordatorio.Text` se actualiza con el mensaje "Recuerda cerrar sesión si terminas". Un `MessageBox` también confirma el inicio de sesión.

Programa 216. Sistema de login con restricción de horarios

Este programa limita el acceso al sistema solo a un rango de horas predefinido, rechazando los intentos de login fuera de ese horario.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtUsuario`.
  - `TextBox`: **Name:** `txtContraseña`, **PasswordChar:** `*`.
  - `Button`: **Name:** `btnLogin`, **Text:** "Iniciar Sesión".
- **Código VB.NET:**

```
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    Dim horaActual As TimeSpan = DateTime.Now.TimeOfDay
    Dim inicio As TimeSpan = New TimeSpan(8, 0, 0)
    Dim fin As TimeSpan = New TimeSpan(18, 0, 0)
    If horaActual >= inicio And horaActual <= fin Then
        MessageBox.Show("Acceso permitido")
    Else
        MessageBox.Show("Acceso fuera de horario permitido")
    End If
End Sub
```

- **Explicación:**
  - Cuando se hace clic en `btnLogin`, el evento `btnLogin_Click` se activa.
  - Se obtiene la `horaActual` del sistema (`DateTime.Now.TimeOfDay`).
  - Se definen dos objetos `TimeSpan`: `inicio` a las 8:00:00 y `fin` a las 18:00:00 (6 PM).
  - Se compara `horaActual` con el rango `inicio` y `fin`. Si está dentro del rango, se muestra "Acceso permitido"; de lo contrario, "Acceso fuera de horario permitido".

Programa 217. Sistema de login con acceso remoto

Este programa es un ejemplo conceptual que indica cómo se gestionaría un acceso remoto, destacando la necesidad de una infraestructura de servidor para una implementación real.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtUsuario`.
    - `TextBox`: **Name:** `txtContraseña`, **PasswordChar:** `*`.
    - `Button`: **Name:** `btnLogin`, **Text:** "Iniciar Sesión".
- **Código VB.NET:**

```
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    MessageBox.Show("Acceso remoto verificado (implementación real
requiere servidor)")
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnLogin`, se muestra un `MessageBox` que indica que el "Acceso remoto" ha sido "verificado" conceptualmente. El mensaje también aclara que una implementación real requeriría una conexión y lógica con un servidor remoto.

Programa 218. Sistema de login con sincronización de datos

Este programa es un ejemplo conceptual que simula la sincronización de datos con un servidor después de un inicio de sesión exitoso.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtUsuario`.
    - `TextBox`: **Name:** `txtContraseña`, **PasswordChar:** `*`.
    - `Button`: **Name:** `btnLogin`, **Text:** "Iniciar Sesión".
- **Código VB.NET:**

```
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    MessageBox.Show("Datos sincronizados con servidor al iniciar sesión")
End Sub
```

- **Explicación:**
    - Cuando se hace clic en `btnLogin`, se muestra un `MessageBox` conceptual que anuncia que "Datos sincronizados con servidor al iniciar sesión". En una aplicación real, este paso implicaría llamadas a servicios web o bases de datos remotas para cargar o actualizar la información del usuario después de la autenticación.

Programa 219. Sistema de login con cifrado de base de datos

Este programa demuestra una forma básica de cifrar una contraseña utilizando Base64 antes de almacenarla (o compararla) en una base de datos.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtUsuario`.
    - `TextBox`: **Name:** `txtContraseña`, **PasswordChar:** `*`.
    - `Button`: **Name:** `btnLogin`, **Text:** "Iniciar Sesión".
- **Código VB.NET:**

```
Imports System.Text
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    Dim contrasenaCifrada As String =
Convert.ToBase64String(Encoding.UTF8.GetBytes(txtContraseña.Text))
    MessageBox.Show("Contraseña cifrada para base de datos: " &
contrasenaCifrada)
End Sub
```

- **Explicación:**
    - Se importa `System.Text` para el manejo de codificaciones.
    - Al hacer clic en `btnLogin`, el texto de `txtContraseña` se convierte a una matriz de bytes usando UTF-8 y luego se codifica a Base64 con `Convert.ToBase64String()`.
    - El `MessageBox` muestra esta `contrasenaCifrada`. Es importante notar que Base64 es una codificación, no un cifrado seguro para contraseñas; para seguridad real, se usarían hashes.

Programa 220. Sistema de login con hash de contraseñas

Este programa demuestra el uso de una función hash (SHA256) para procesar una contraseña, lo cual es una práctica segura para almacenar contraseñas en bases de datos.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtUsuario`.
  - `TextBox`: **Name:** `txtContraseña`, **PasswordChar:** `*`.
  - `Button`: **Name:** `btnLogin`, **Text:** "Iniciar Sesión".
- **Código VB.NET:**

```
Imports System.Security.Cryptography
Imports System.Text
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    Dim sha As SHA256 = SHA256.Create()
    Dim bytes As Byte() = Encoding.UTF8.GetBytes(txtContraseña.Text)
    Dim hash As Byte() = sha.ComputeHash(bytes)
    Dim hashString As String = BitConverter.ToString(hash).Replace("-",
"")
    MessageBox.Show("Hash de contraseña: " & hashString)
End Sub
```

- **Explicación:**
  - Se importan `System.Security.Cryptography` y `System.Text`.
  - Al hacer clic en `btnLogin`:
    - Se crea una instancia de `SHA256`.
    - El texto de `txtContraseña` se convierte a bytes UTF-8.
    - `sha.ComputeHash(bytes)` calcula el valor hash SHA256 de los bytes.
    - `BitConverter.ToString(hash).Replace("-", "")` convierte la matriz de bytes hash a una representación hexadecimal legible sin guiones.
    - El `MessageBox` muestra este `hashString`. Este es el método recomendado para almacenar contraseñas de forma segura (junto con un "salt").

Programa 221. Sistema de inventario básico

Este programa implementa un sistema de inventario simple que permite agregar productos con su cantidad a un `ListBox`.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtProducto`.
  - `TextBox`: **Name:** `txtCantidad`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar".
  - `ListBox`: **Name:** `lstInventario`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstInventario.Items.Add(txtProducto.Text & " - Cantidad: " &
txtCantidad.Text)
    txtProducto.Clear()
    txtCantidad.Clear()
End Sub
```

- **Explicación:**
  - Cuando se hace clic en `btnAgregar`, el evento `btnAgregar_Click` se activa.
  - El texto de `txtProducto` y `txtCantidad` se concatena en una cadena "Producto - Cantidad: X" y se añade como un nuevo elemento a `lstInventario`.
  - Los `TextBox` `txtProducto` y `txtCantidad` se limpian para la próxima entrada.

Programa 222. Sistema de inventario con base de datos

Este programa utiliza un `DataTable` como simulación de una base de datos para almacenar productos y sus cantidades en un `DataGridView`.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtProducto`.
    - `TextBox`: **Name:** `txtCantidad`.
    - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar".
    - `DataGridView`: **Name:** `dgvInventario`.
- **Código VB.NET:**

```
Dim dt As New DataTable()
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
    dt.Columns.Add("Producto")
    dt.Columns.Add("Cantidad")
    dgvInventario.DataSource = dt
End Sub
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    dt.Rows.Add(txtProducto.Text, txtCantidad.Text)
    txtProducto.Clear()
    txtCantidad.Clear()
End Sub
```

- **Explicación:**
    - Un `DataTable` llamado `dt` se declara a nivel de clase.
    - En `Form1_Load`, se añaden dos columnas al `DataTable`: "Producto" y "Cantidad". Luego, `dgvInventario.DataSource` se enlaza a `dt`, lo que permite que el `DataGridView` muestre los datos del `DataTable`.
    - Cuando se hace clic en `btnAgregar`, una nueva fila con el `txtProducto.Text` y `txtCantidad.Text` se añade al `dt.Rows`. Los `TextBox` se limpian.

Programa 223. Sistema de inventario con facturación

Este programa calcula el total de un producto (cantidad * precio) y lo añade a una lista de facturación.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtProducto`.
  - `TextBox`: **Name:** `txtCantidad`.
  - `TextBox`: **Name:** `txtPrecio`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar".
  - `ListBox`: **Name:** `lstFactura`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    Dim total As Double = Val(txtCantidad.Text) * Val(txtPrecio.Text)
    lstFactura.Items.Add(txtProducto.Text & " - Cantidad: " &
txtCantidad.Text & " - Total: " & total)
    txtProducto.Clear()
    txtCantidad.Clear()
    txtPrecio.Clear()
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnAgregar`, el evento `btnAgregar_Click` se activa.
  - Se calcula la variable `total` multiplicando el valor numérico de `txtCantidad.Text` por `txtPrecio.Text`.
  - Una cadena que incluye el nombre del producto, la cantidad y el total se añade al `lstFactura`.
  - Los `TextBox` `txtProducto`, `txtCantidad` y `txtPrecio` se limpian.

Programa 224. Sistema de inventario con reportes

Este programa genera un reporte simple del inventario, mostrando todos los elementos listados en el `ListBox` en un `MessageBox`.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnReporte`, **Text:** "Generar Reporte".
  - `ListBox`: **Name:** `lstInventario`.
- **Código VB.NET:**

```
Private Sub btnReporte_Click(sender As Object, e As EventArgs) Handles
btnReporte.Click
    Dim reporte As String = "Reporte Inventario:" & Environment.NewLine
    For Each item In lstInventario.Items
        reporte &= item.ToString() & Environment.NewLine
    Next
    MessageBox.Show(reporte)
End Sub
```

- **Explicación:**
  - Cuando se hace clic en `btnReporte`, el evento `btnReporte_Click` se ejecuta.
  - Se inicializa una cadena `reporte` con el título "Reporte Inventario:" seguido de un salto de línea.
  - Un bucle `For Each` itera sobre cada `item` en `lstInventario.Items`, concatenando cada elemento (convertido a cadena) con un salto de línea a la cadena `reporte`.
  - Finalmente, el `reporte` completo se muestra en un `MessageBox`.

Programa 225. Sistema de inventario con gráficos

Este programa muestra un gráfico de columnas en un control `Chart` con datos de inventario simulados.

- **Controles necesarios en el formulario:**
  - `Chart`: **Name:** `chartInventario`, **ChartType:** `Column`.
  - `Button`: **Name:** `btnGraficar`, **Text:** "Mostrar Gráfico".
- **Requisito adicional:**
  - Asegurarse de que el control `Chart` esté disponible en el `Toolbox` (puede requerir añadir la referencia `System.Windows.Forms.DataVisualization` al proyecto).
- **Código VB.NET:**

```
Private Sub btnGraficar_Click(sender As Object, e As EventArgs) Handles
btnGraficar.Click
    chartInventario.Series.Clear()
    Dim serie As New DataVisualization.Charting.Series("Inventario")
    serie.ChartType = DataVisualization.Charting.SeriesChartType.Column
    serie.Points.AddXY("Producto A", 10)
    serie.Points.AddXY("Producto B", 15)
    chartInventario.Series.Add(serie)
End Sub
```

- **Explicación:**
  - Cuando se hace clic en `btnGraficar`, el evento `btnGraficar_Click` se activa.
  - `chartInventario.Series.Clear()` limpia cualquier serie de datos existente en el gráfico.
  - Se crea una nueva serie llamada "Inventario" y su `ChartType` se establece en `Column` para un gráfico de barras.
  - Se añaden puntos de datos simulados (`"Producto A"`, `10`) y (`"Producto B"`, `15`) a la serie con `serie.Points.AddXY()`.
  - Finalmente, la `serie` se añade a `chartInventario.Series`, lo que hace que el gráfico se muestre.

Programa 226. Sistema de inventario con exportación Excel

Este programa conceptual simula la exportación de datos de inventario a una hoja de cálculo de Microsoft Excel.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnExportar`, **Text:** "Exportar a Excel".
- **Requisito adicional:**
    - Este programa es conceptual y requeriría agregar una referencia a la **biblioteca de objetos de Microsoft Excel** en el proyecto para una implementación real.
- **Código VB.NET:**

```
Imports Microsoft.Office.Interop.Excel
Private Sub btnExportar_Click(sender As Object, e As EventArgs) Handles
btnExportar.Click
    MessageBox.Show("Datos exportados a Excel (requiere referencia a
Microsoft Excel)")
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnExportar`, se muestra un `MessageBox` conceptual indicando que los datos se han "exportado a Excel" y recordando la necesidad de la referencia a la librería de Excel.

Programa 227. Sistema de inventario con importación Excel

Este programa conceptual simula la importación de datos de inventario desde una hoja de cálculo de Microsoft Excel.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnImportar`, **Text:** "Importar desde Excel".
- **Requisito adicional:**
    - Este programa es conceptual y requeriría agregar una referencia a la **biblioteca de objetos de Microsoft Excel** en el proyecto para una implementación real.
- **Código VB.NET:**

```
Imports Microsoft.Office.Interop.Excel
Private Sub btnImportar_Click(sender As Object, e As EventArgs) Handles
btnImportar.Click
    MessageBox.Show("Datos importados desde Excel (requiere referencia a
Microsoft Excel)")
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnImportar`, se muestra un `MessageBox` conceptual que indica que los datos se han "importado desde Excel" y recuerda la necesidad de la referencia a la librería de Excel.

Programa 228. Sistema de inventario con códigos de barra

Este programa permite agregar códigos de barra (simplemente como texto) a una lista del inventario.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtCodigo`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Código".
  - `ListBox`: **Name:** `lstCodigos`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstCodigos.Items.Add(txtCodigo.Text)
    txtCodigo.Clear()
End Sub
```

- **Explicación:**
  - Cuando se hace clic en `btnAgregar`, el texto del `txtCodigo` se añade como un nuevo elemento a `lstCodigos`.
  - El `txtCodigo` se limpia después de agregar el código.

Programa 229. Sistema de inventario con múltiples usuarios

Este programa simula el registro de acciones de inventario realizadas por diferentes tipos de usuarios seleccionados de un `ComboBox`.

- **Controles necesarios en el formulario:**
  - `ComboBox`: **Name:** `cmbUsuarios`, **Items:** "Admin", "Vendedor".
  - `TextBox`: **Name:** `txtProducto`.
  - `TextBox`: **Name:** `txtCantidad`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar".
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    Dim usuario As String = cmbUsuarios.SelectedItem
    MessageBox.Show("Usuario: " & usuario & " agregó " & txtProducto.Text
& " Cantidad: " & txtCantidad.Text)
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnAgregar`, se obtiene el usuario seleccionado del `cmbUsuarios.SelectedItem`.
  - Un `MessageBox` muestra un mensaje indicando qué usuario agregó qué producto y cantidad, simulando una acción realizada por un usuario específico.

Programa 230. Sistema de inventario con permisos

Este programa demuestra cómo la acción de "eliminar producto" puede estar restringida a ciertos roles, como "Admin".

- **Controles necesarios en el formulario:**
  - `ComboBox`: **Name:** `cmbRoles`, **Items:** "Admin", "Vendedor".
  - `Button`: **Name:** `btnEliminar`, **Text:** "Eliminar Producto".
- **Código VB.NET:**

```
Private Sub btnEliminar_Click(sender As Object, e As EventArgs) Handles
btnEliminar.Click
    If cmbRoles.SelectedItem = "Admin" Then
        MessageBox.Show("Producto eliminado")
    Else
        MessageBox.Show("No tiene permisos para eliminar")
    End If
End Sub
```

- **Explicación:**
  - Cuando se hace clic en `btnEliminar`, el evento `btnEliminar_Click` se activa.
  - El programa verifica si el rol seleccionado en `cmbRoles.SelectedItem` es "Admin".
  - Si es "Admin", se muestra "Producto eliminado"; de lo contrario, se muestra "No tiene permisos para eliminar".

Programa 231. Sistema de inventario con auditoría

Este programa registra las adiciones de productos al inventario en un `ListBox` de auditoría, con la fecha y hora del evento.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtProducto`.
  - `TextBox`: **Name:** `txtCantidad`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar".
  - `ListBox`: **Name:** `lstInventario`.
  - `ListBox`: **Name:** `lstAuditoria`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstInventario.Items.Add(txtProducto.Text & " - Cantidad: " &
txtCantidad.Text)
    lstAuditoria.Items.Add(DateTime.Now.ToString() & " - Producto
agregado: " & txtProducto.Text)
    txtProducto.Clear()
    txtCantidad.Clear()
End Sub
```

- **Explicación:**
  - Cuando se hace clic en `btnAgregar`, se añade el producto y su cantidad a `lstInventario`.
  - Adicionalmente, se registra una entrada en `lstAuditoria` con la fecha y hora actuales, y el nombre del producto agregado.
  - Los `TextBox` se limpian.

Programa 232. Sistema de inventario con estadísticas

Este programa muestra estadísticas básicas del inventario, como el número total de productos, en un `MessageBox`.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnEstadisticas`, **Text:** "Mostrar Estadísticas".
  - `ListBox`: **Name:** `lstInventario`.
- **Código VB.NET:**

```
Private Sub btnEstadisticas_Click(sender As Object, e As EventArgs)
Handles btnEstadisticas.Click
    Dim totalProductos As Integer = lstInventario.Items.Count
    MessageBox.Show("Total de productos en inventario: " & totalProductos)
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnEstadisticas`, se obtiene el número total de elementos en `lstInventario.Items.Count`.
  - Un `MessageBox` muestra este `totalProductos` como una estadística simple del inventario.

Programa 233. Sistema de inventario con gráficos dinámicos

Este programa actualiza un gráfico de columnas en un control `Chart` para visualizar los productos en el `lstInventario`.

- **Controles necesarios en el formulario:**
  - `Chart`: **Name:** `chartInventario`, **ChartType:** `Column`.
  - `Button`: **Name:** `btnActualizarGrafico`, **Text:** "Actualizar Gráfico".
  - `ListBox`: **Name:** `lstInventario`.
- **Requisito adicional:**
  - Asegurarse de que el control `Chart` esté disponible en el `Toolbox` (puede requerir añadir la referencia `System.Windows.Forms.DataVisualization` al proyecto).
- **Código VB.NET:**

```
Private Sub btnActualizarGrafico_Click(sender As Object, e As EventArgs)
Handles btnActualizarGrafico.Click
    chartInventario.Series.Clear()
    Dim serie As New DataVisualization.Charting.Series("Inventario")
    serie.ChartType = DataVisualization.Charting.SeriesChartType.Column
    For Each item In lstInventario.Items
        serie.Points.AddXY(item.ToString(), 1)
    Next
    chartInventario.Series.Add(serie)
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnActualizarGrafico`, se limpia el `chartInventario.Series`.
  - Se crea una nueva `serie` de tipo columna. Un bucle `For Each` itera sobre cada `item` en `lstInventario.Items`.
  - Para cada ítem, se añade un punto al gráfico con el nombre del ítem y un valor de 1, representando la existencia de cada producto. Esto visualiza la cantidad de elementos distintos en la lista.
  - La serie se añade al gráfico para su visualización.

Programa 234. Sistema de inventario con múltiples almacenes

Este programa permite al usuario agregar productos a un inventario, especificando el almacén de origen mediante un `ComboBox`.

- **Controles necesarios en el formulario:**
    - `ComboBox`: **Name:** `cmbAlmacenes`, **Items:** "Almacén 1", "Almacén 2".
    - `TextBox`: **Name:** `txtProducto`.
    - `TextBox`: **Name:** `txtCantidad`.
    - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar".
    - `ListBox`: **Name:** `lstInventario`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    Dim almacen As String = cmbAlmacenes.SelectedItem
    lstInventario.Items.Add(almacen & " - " & txtProducto.Text & " -
Cantidad: " & txtCantidad.Text)
    txtProducto.Clear()
    txtCantidad.Clear()
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnAgregar`, se obtiene el `almacen` seleccionado de `cmbAlmacenes`.
    - El producto, la cantidad y el almacén se combinan en una cadena y se añaden a `lstInventario`.
    - Los `TextBox` `txtProducto` y `txtCantidad` se limpian.

Programa 235. Sistema de inventario con conexión remota

Este programa es un ejemplo conceptual que simula la conexión a un servidor remoto para la gestión del inventario.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnConectar`, **Text:** "Conectar a Servidor".
- **Código VB.NET:**

```
Private Sub btnConectar_Click(sender As Object, e As EventArgs) Handles
btnConectar.Click
    MessageBox.Show("Conexión remota establecida (simulada)")
End Sub
```

- **Explicación:**
    - Cuando se hace clic en `btnConectar`, se muestra un `MessageBox` que indica conceptualmente que se ha establecido una "Conexión remota". El mensaje aclara que esto es una simulación y que una implementación real requeriría código de red complejo.

Programa 236. Sistema de inventario con backup automático

Este programa conceptual simula la realización de un backup automático de los datos del inventario.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnBackup`, **Text:** "Realizar Backup".
- **Código VB.NET:**

```
Private Sub btnBackup_Click(sender As Object, e As EventArgs) Handles
btnBackup.Click
    MessageBox.Show("Backup automático realizado correctamente")
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnBackup`, se muestra un `MessageBox` conceptual que confirma que el "Backup automático" ha sido "realizado correctamente". Una implementación real implicaría copiar archivos de base de datos, exportar datos, o usar herramientas de backup.

Programa 237. Sistema de inventario con restauración de datos

Este programa conceptual simula la restauración de datos del inventario a partir de un backup.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnRestaurar`, **Text:** "Restaurar Backup".
- **Código VB.NET:**

```
Private Sub btnRestaurar_Click(sender As Object, e As EventArgs) Handles
btnRestaurar.Click
    MessageBox.Show("Datos restaurados desde backup")
End Sub
```

- **Explicación:**
    - Cuando se hace clic en `btnRestaurar`, se muestra un `MessageBox` que indica conceptualmente que los "Datos" han sido "restaurados desde backup". Una implementación real implicaría reemplazar archivos de base de datos o importar datos de un backup anterior.

Programa 238. Sistema de inventario con notificaciones

Este programa muestra una notificación en el área de la bandeja del sistema (system tray) para alertar al usuario sobre eventos importantes, como el bajo stock de productos.

- **Controles necesarios en el formulario:**
  - `NotifyIcon`: **Name:** `NotifyIcon1`, **Icon:** `icono.ico`.
  - `Button`: **Name:** `btnNotificar`, **Text:** "Notificar Stock Bajo".
- **Requisito adicional:**
  - Se requiere un archivo de ícono (`icono.ico`) para el `NotifyIcon`.
- **Código VB.NET:**

```
Private Sub btnNotificar_Click(sender As Object, e As EventArgs) Handles
btnNotificar.Click
    NotifyIcon1.Visible = True
    NotifyIcon1.ShowBalloonTip(3000, "Inventario", "Alerta: Stock bajo en
algunos productos", ToolTipIcon.Warning)
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnNotificar`, `NotifyIcon1.Visible` se establece en `True` para asegurar que el ícono de la aplicación sea visible en la bandeja del sistema.
  - Luego, `NotifyIcon1.ShowBalloonTip()` muestra una notificación en forma de globo con una duración de 3 segundos (3000 ms), el título "Inventario", el mensaje "Alerta: Stock bajo en algunos productos" y un `ToolTipIcon.Warning` para indicar una alerta.

Programa 239. Sistema de inventario con control de lotes

Este programa permite agregar productos al inventario especificando también un número de lote junto con el producto y la cantidad.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtProducto`.
  - `TextBox`: **Name:** `txtLote`.
  - `TextBox`: **Name:** `txtCantidad`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Lote".
  - `ListBox`: **Name:** `lstInventario`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstInventario.Items.Add("Producto: " & txtProducto.Text & " - Lote: "
& txtLote.Text & " - Cantidad: " & txtCantidad.Text)
    txtProducto.Clear()
    txtLote.Clear()
    txtCantidad.Clear()
End Sub
```

- **Explicación:**
  - Cuando se hace clic en `btnAgregar`, el texto de `txtProducto`, `txtLote` y `txtCantidad` se combinan en una cadena que incluye la información del producto, lote y cantidad. Esta cadena se añade como un nuevo elemento a `lstInventario`.
  - Los tres `TextBox` se limpian para la siguiente entrada.

Programa 240. Sistema de inventario con caducidades

Este programa permite agregar productos al inventario junto con una fecha de caducidad, seleccionada por el usuario, y la cantidad.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtProducto`.
  - `DateTimePicker`: **Name:** `dtpCaducidad`.
  - `TextBox`: **Name:** `txtCantidad`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Producto".
  - `ListBox`: **Name:** `lstInventario`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstInventario.Items.Add("Producto: " & txtProducto.Text & " -
Cantidad: " & txtCantidad.Text & " - Caduca: " &
dtpCaducidad.Value.ToShortDateString())
    txtProducto.Clear()
    txtCantidad.Clear()
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnAgregar`, el texto de `txtProducto` y `txtCantidad`, junto con la fecha de caducidad seleccionada en `dtpCaducidad.Value` (convertida a formato de cadena corta), se combinan en una cadena. Esta cadena se añade como un nuevo elemento a `lstInventario`.
  - Los `TextBox` `txtProducto` y `txtCantidad` se limpian para la próxima entrada.

Programa 241. Sistema de ventas básico

Este programa implementa un sistema de ventas simple que permite agregar productos con su cantidad a una lista de ventas.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtProducto`.
  - `TextBox`: **Name:** `txtCantidad`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Venta".
  - `ListBox`: **Name:** `lstVentas`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstVentas.Items.Add(txtProducto.Text & " - Cantidad: " &
txtCantidad.Text)
    txtProducto.Clear()
    txtCantidad.Clear()
End Sub
```

- **Explicación:**
  - Cuando se hace clic en `btnAgregar`, el evento `btnAgregar_Click` se activa.
  - El texto de `txtProducto` y `txtCantidad` se concatena en una cadena que incluye el nombre del producto y la cantidad, la cual se añade como un nuevo elemento a `lstVentas`.
  - Los `TextBox` `txtProducto` y `txtCantidad` se limpian después de la operación.

Programa 242. Sistema de ventas con base de datos

Este programa utiliza un `DataTable` como simulación de una base de datos para almacenar productos y sus cantidades en un `DataGridView`.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtProducto`.
    - `TextBox`: **Name:** `txtCantidad`.
    - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Venta".
    - `DataGridView`: **Name:** `dgvVentas`.
- **Código VB.NET:**

```
Dim dt As New DataTable()
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
    dt.Columns.Add("Producto")
    dt.Columns.Add("Cantidad")
    dgvVentas.DataSource = dt
End Sub
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    dt.Rows.Add(txtProducto.Text, txtCantidad.Text)
    txtProducto.Clear()
    txtCantidad.Clear()
End Sub
```

- **Explicación:**
    - Un `DataTable` llamado `dt` se declara a nivel de clase.
    - En `Form1_Load`, se añaden dos columnas al `DataTable`: "Producto" y "Cantidad". Luego, `dgvVentas.DataSource` se enlaza a `dt`, lo que permite que el `DataGridView` muestre los datos del `DataTable`.
    - Cuando se hace clic en `btnAgregar`, una nueva fila con el texto de `txtProducto` y `txtCantidad` se añade a `dt.Rows`. Los `TextBox` se limpian.

Programa 243. Sistema de ventas con clientes

Este programa permite asociar ventas a diferentes clientes seleccionados de un `ComboBox`.

- **Controles necesarios en el formulario:**
  - `ComboBox`: **Name:** `cmbClientes`, **Items:** "Cliente 1", "Cliente 2".
  - `TextBox`: **Name:** `txtProducto`.
  - `TextBox`: **Name:** `txtCantidad`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Venta".
  - `ListBox`: **Name:** `lstVentas`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstVentas.Items.Add("Cliente: " & cmbClientes.SelectedItem & " -
Producto: " & txtProducto.Text & " - Cantidad: " & txtCantidad.Text)
    txtProducto.Clear()
    txtCantidad.Clear()
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnAgregar`, el evento `btnAgregar_Click` se activa.
  - El programa obtiene el cliente seleccionado de `cmbClientes.SelectedItem`.
  - Una cadena que incluye el cliente, el producto y la cantidad se añade a `lstVentas`.
  - Los `TextBox` `txtProducto` y `txtCantidad` se limpian.

Programa 244. Sistema de ventas con facturación

Este programa calcula el total de una venta (cantidad * precio) y lo añade a una lista de facturación.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtProducto`.
  - `TextBox`: **Name:** `txtCantidad`.
  - `TextBox`: **Name:** `txtPrecio`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Factura".
  - `ListBox`: **Name:** `lstFactura`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    Dim total As Double = Val(txtCantidad.Text) * Val(txtPrecio.Text)
    lstFactura.Items.Add(txtProducto.Text & " - Cantidad: " &
txtCantidad.Text & " - Total: " & total)
    txtProducto.Clear()
    txtCantidad.Clear()
    txtPrecio.Clear()
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnAgregar`, el evento `btnAgregar_Click` se ejecuta.
  - Se calcula la variable `total` multiplicando el valor numérico de `txtCantidad.Text` por `txtPrecio.Text`.
  - Una cadena que incluye el nombre del producto, la cantidad y el total se añade al `lstFactura`.
  - Los `TextBox` `txtProducto`, `txtCantidad` y `txtPrecio` se limpian.

Programa 245. Sistema de ventas con reportes

Este programa genera un reporte simple de las ventas, mostrando todos los elementos listados en el `ListBox` en un `MessageBox`.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnReporte`, **Text:** "Generar Reporte".
  - `ListBox`: **Name:** `lstVentas`.
- **Código VB.NET:**

```
Private Sub btnReporte_Click(sender As Object, e As EventArgs) Handles
btnReporte.Click
    Dim reporte As String = "Reporte de Ventas:" & Environment.NewLine
    For Each item In lstVentas.Items
        reporte &= item.ToString() & Environment.NewLine
    Next
    MessageBox.Show(reporte)
End Sub
```

- **Explicación:**
  - Cuando se hace clic en `btnReporte`, el evento `btnReporte_Click` se ejecuta.
  - Se inicializa una cadena `reporte` con el título "Reporte de Ventas:" seguido de un salto de línea.
  - Un bucle `For Each` itera sobre cada `item` en `lstVentas.Items`, concatenando cada elemento (convertido a cadena) con un salto de línea a la cadena `reporte`.
  - Finalmente, el `reporte` completo se muestra en un `MessageBox`.

Programa 246. Sistema de ventas con estadísticas

Este programa muestra estadísticas básicas de las ventas, como el número total de ventas registradas, en un `MessageBox`.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnEstadisticas`, **Text:** "Mostrar Estadísticas".
  - `ListBox`: **Name:** `lstVentas`.
- **Código VB.NET:**

```
Private Sub btnEstadisticas_Click(sender As Object, e As EventArgs)
Handles btnEstadisticas.Click
    Dim totalVentas As Integer = lstVentas.Items.Count
    MessageBox.Show("Total de ventas registradas: " & totalVentas)
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnEstadisticas`, el evento `btnEstadisticas_Click` se activa.
  - `lstVentas.Items.Count` obtiene el número total de elementos (ventas) actualmente en el `ListBox`.
  - Un `MessageBox` muestra este `totalVentas` como una estadística simple.

Programa 247. Sistema de ventas con gráficos

Este programa muestra un gráfico de columnas en un control `Chart` para visualizar las ventas.

- **Controles necesarios en el formulario:**
    - `Chart`: **Name:** `chartVentas`, **ChartType:** `Column`.
    - `Button`: **Name:** `btnGraficar`, **Text:** "Mostrar Gráfico".
    - `ListBox`: **Name:** `lstVentas`.
- **Requisito adicional:**
    - Asegurarse de que el control `Chart` esté disponible en el `Toolbox` (puede requerir añadir la referencia `System.Windows.Forms.DataVisualization` al proyecto).
- **Código VB.NET:**

```
Private Sub btnGraficar_Click(sender As Object, e As EventArgs) Handles
btnGraficar.Click
    chartVentas.Series.Clear()
    Dim serie As New DataVisualization.Charting.Series("Ventas")
    serie.ChartType = DataVisualization.Charting.SeriesChartType.Column
    For Each item In lstVentas.Items
        serie.Points.AddXY(item.ToString(), 1)
    Next
    chartVentas.Series.Add(serie)
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnGraficar`, el evento `btnGraficar_Click` se activa.
    - `chartVentas.Series.Clear()` limpia cualquier serie de datos existente en el gráfico.
    - Se crea una nueva serie llamada "Ventas" y su `ChartType` se establece en `Column`.
    - Un bucle `For Each` itera sobre cada `item` en `lstVentas.Items`.
    - Para cada ítem, se añade un punto al gráfico con el nombre del ítem y un valor de 1, representando la existencia de cada venta.
    - Finalmente, la `serie` se añade a `chartVentas.Series`, lo que hace que el gráfico se muestre.

Programa 248. Sistema de ventas con exportación Excel

Este programa conceptual simula la exportación de datos de ventas a una hoja de cálculo de Microsoft Excel.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnExportar`, **Text:** "Exportar a Excel".
- **Requisito adicional:**
    - Este programa es conceptual y requeriría agregar una referencia a la **biblioteca de objetos de Microsoft Excel** en el proyecto para una implementación real.
- **Código VB.NET:**

```
Imports Microsoft.Office.Interop.Excel
Private Sub btnExportar_Click(sender As Object, e As EventArgs) Handles
btnExportar.Click
    MessageBox.Show("Ventas exportadas a Excel (requiere referencia a
Microsoft Excel)")
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnExportar`, se muestra un `MessageBox` conceptual que indica que las "Ventas" han sido "exportadas a Excel" y recuerda la necesidad de la referencia a la librería de Excel.

Programa 249. Sistema de ventas con importación Excel

Este programa conceptual simula la importación de datos de ventas desde una hoja de cálculo de Microsoft Excel.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnImportar`, **Text:** "Importar desde Excel".
- **Requisito adicional:**
    - Este programa es conceptual y requeriría agregar una referencia a la **biblioteca de objetos de Microsoft Excel** en el proyecto para una implementación real.
- **Código VB.NET:**

```
Imports Microsoft.Office.Interop.Excel
Private Sub btnImportar_Click(sender As Object, e As EventArgs) Handles
btnImportar.Click
    MessageBox.Show("Ventas importadas desde Excel (requiere referencia a
Microsoft Excel)")
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnImportar`, se muestra un `MessageBox` conceptual que indica que las "Ventas" han sido "importadas desde Excel" y recuerda la necesidad de la referencia a la librería de Excel.

Programa 250. Sistema de ventas con múltiples usuarios

Este programa simula el registro de ventas realizadas por diferentes tipos de usuarios seleccionados de un `ComboBox`.

- **Controles necesarios en el formulario:**
  - `ComboBox`: **Name:** `cmbUsuarios`, **Items:** "Admin", "Vendedor".
  - `TextBox`: **Name:** `txtProducto`.
  - `TextBox`: **Name:** `txtCantidad`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Venta".
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    Dim usuario As String = cmbUsuarios.SelectedItem
    MessageBox.Show("Usuario: " & usuario & " agregó Producto: " &
txtProducto.Text & " Cantidad: " & txtCantidad.Text)
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnAgregar`, se obtiene el usuario seleccionado de `cmbUsuarios.SelectedItem`.
  - Un `MessageBox` muestra un mensaje indicando qué usuario agregó qué producto y cantidad, simulando una acción realizada por un usuario específico.

Programa 251. Sistema de ventas con permisos

Este programa demuestra cómo la acción de "eliminar venta" puede estar restringida a ciertos roles, como "Admin".

- **Controles necesarios en el formulario:**
  - `ComboBox`: **Name:** `cmbRoles`, **Items:** "Admin", "Vendedor".
  - `Button`: **Name:** `btnEliminar`, **Text:** "Eliminar Venta".
- **Código VB.NET:**

```
Private Sub btnEliminar_Click(sender As Object, e As EventArgs) Handles
btnEliminar.Click
    If cmbRoles.SelectedItem = "Admin" Then
        MessageBox.Show("Venta eliminada")
    Else
        MessageBox.Show("No tiene permisos para eliminar")
    End If
End Sub
```

- **Explicación:**
  - Cuando se hace clic en `btnEliminar`, el evento `btnEliminar_Click` se activa.
  - El programa verifica si el rol seleccionado en `cmbRoles.SelectedItem` es "Admin".
  - Si es "Admin", se muestra "Venta eliminada"; de lo contrario, se muestra "No tiene permisos para eliminar".

Programa 252. Sistema de ventas con auditoría

Este programa registra las adiciones de ventas en un `ListBox` de auditoría, con la fecha y hora del evento.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtProducto`.
  - `TextBox`: **Name:** `txtCantidad`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Venta".
  - `ListBox`: **Name:** `lstVentas`.
  - `ListBox`: **Name:** `lstAuditoria`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstVentas.Items.Add(txtProducto.Text & " - Cantidad: " &
txtCantidad.Text)
    lstAuditoria.Items.Add(DateTime.Now.ToString() & " - Venta agregada: "
& txtProducto.Text)
    txtProducto.Clear()
    txtCantidad.Clear()
End Sub
```

- **Explicación:**
  - Cuando se hace clic en `btnAgregar`, el producto y su cantidad se añaden a `lstVentas`.
  - Adicionalmente, se registra una entrada en `lstAuditoria` con la fecha y hora actuales, y el nombre del producto vendido.
  - Los `TextBox` se limpian.

Programa 253. Sistema de ventas con conexión remota

Este programa es un ejemplo conceptual que simula la conexión a un servidor remoto para la gestión de ventas.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnConectar`, **Text:** "Conectar a Servidor".
- **Código VB.NET:**

```
Private Sub btnConectar_Click(sender As Object, e As EventArgs) Handles
btnConectar.Click
    MessageBox.Show("Conexión remota establecida (simulada)")
End Sub
```

- **Explicación:**
    - Cuando se hace clic en `btnConectar`, se muestra un `MessageBox` que indica conceptualmente que se ha establecido una "Conexión remota". El mensaje aclara que esto es una simulación y que una implementación real requeriría código de red complejo.

Programa 254. Sistema de ventas con backup automático

Este programa conceptual simula la realización de un backup automático de los datos de ventas.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnBackup`, **Text:** "Realizar Backup".
- **Código VB.NET:**

```
Private Sub btnBackup_Click(sender As Object, e As EventArgs) Handles
btnBackup.Click
    MessageBox.Show("Backup automático de ventas realizado correctamente")
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnBackup`, se muestra un `MessageBox` conceptual que confirma que el "Backup automático de ventas" ha sido "realizado correctamente". Una implementación real implicaría copiar archivos de base de datos, exportar datos, o usar herramientas de backup.

Programa 255. Sistema de ventas con restauración de datos

Este programa conceptual simula la restauración de datos de ventas a partir de un backup.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnRestaurar`, **Text:** "Restaurar Backup".
- **Código VB.NET:**

```
Private Sub btnRestaurar_Click(sender As Object, e As EventArgs) Handles
btnRestaurar.Click
    MessageBox.Show("Ventas restauradas desde backup")
End Sub
```

- **Explicación:**
    - Cuando se hace clic en `btnRestaurar`, se muestra un `MessageBox` que indica conceptualmente que las "Ventas" han sido "restauradas desde backup". Una implementación real implicaría reemplazar archivos de base de datos o importar datos de un backup anterior.

Programa 256. Sistema de ventas con notificaciones

Este programa muestra una notificación en el área de la bandeja del sistema (system tray) para alertar al usuario sobre una nueva venta registrada.

- **Controles necesarios en el formulario:**
  - `NotifyIcon`: **Name:** `NotifyIcon1`, **Icon:** `icono.ico`.
  - `Button`: **Name:** `btnNotificar`, **Text:** "Notificar Venta".
- **Requisito adicional:**
  - Se requiere un archivo de ícono (`icono.ico`) para el `NotifyIcon`.
- **Código VB.NET:**

```
Private Sub btnNotificar_Click(sender As Object, e As EventArgs) Handles
btnNotificar.Click
    NotifyIcon1.Visible = True
    NotifyIcon1.ShowBalloonTip(3000, "Ventas", "Nueva venta registrada",
ToolTipIcon.Info)
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnNotificar`, `NotifyIcon1.Visible` se establece en `True` para asegurar que el ícono de la aplicación sea visible en la bandeja del sistema.
  - Luego, `NotifyIcon1.ShowBalloonTip()` muestra una notificación en forma de globo con una duración de 3 segundos (3000 ms), el título "Ventas", el mensaje "Nueva venta registrada" y un `ToolTipIcon.Info` para indicar información.

Programa 257. Sistema de ventas con control de stock

Este programa simula la gestión de stock, decrementando la cantidad de un producto si la venta es posible, o mostrando una alerta si el stock es insuficiente.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtProducto`.
    - `TextBox`: **Name:** `txtCantidad`.
    - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Venta".
- **Código VB.NET:**

```
Dim stock As New Dictionary(Of String, Integer) From {{"ProductoA", 10}, 
{"ProductoB", 5}}
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles 
btnAgregar.Click
    Dim producto As String = txtProducto.Text
    Dim cantidad As Integer = Val(txtCantidad.Text)
    If stock.ContainsKey(producto) AndAlso stock(producto) >= cantidad 
Then
        stock(producto) -= cantidad
        MessageBox.Show("Venta realizada. Stock restante: " & 
stock(producto))
    Else
        MessageBox.Show("Stock insuficiente")
    End If
End Sub
```

- **Explicación:**
    - Un `Dictionary` llamado `stock` se declara a nivel de clase, inicializado con algunos productos y sus cantidades.
    - Cuando se hace clic en `btnAgregar`, el producto y la cantidad se obtienen de los `TextBox`.
    - El programa verifica si el `stock` contiene el `producto` y si la `cantidad` a vender es menor o igual al stock disponible.
    - Si la venta es posible, el `stock` del producto se decrementa, y se muestra un `MessageBox` con el stock restante. Si el stock es insuficiente, se muestra un mensaje de "Stock insuficiente".

Programa 258. Sistema de ventas con caducidades

Este programa permite registrar ventas junto con la fecha de caducidad del producto vendido.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtProducto`.
    - `DateTimePicker`: **Name:** `dtpCaducidad`.
    - `TextBox`: **Name:** `txtCantidad`.
    - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Venta".
    - `ListBox`: **Name:** `lstVentas`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstVentas.Items.Add("Producto: " & txtProducto.Text & " - Cantidad: "
& txtCantidad.Text & " - Caduca: " &
dtpCaducidad.Value.ToShortDateString())
    txtProducto.Clear()
    txtCantidad.Clear()
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnAgregar`, el texto de `txtProducto` y `txtCantidad`, junto con la fecha de caducidad seleccionada en `dtpCaducidad.Value` (convertida a formato de cadena corta), se combinan en una cadena.
    - Esta cadena se añade como un nuevo elemento a `lstVentas`.
    - Los `TextBox` `txtProducto` y `txtCantidad` se limpian.

Programa 259. Sistema de ventas con código de barras

Este programa permite al usuario ingresar un código de barras para registrar una venta.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtCodigo`.
    - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Venta".
    - `ListBox`: **Name:** `lstVentas`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstVentas.Items.Add("Código: " & txtCodigo.Text)
    txtCodigo.Clear()
End Sub
```

- **Explicación:**
    - Cuando se hace clic en `btnAgregar`, el texto del `txtCodigo` se añade como un nuevo elemento a `lstVentas`.
    - El `txtCodigo` se limpia después de agregar el código.

Programa 260. Sistema de ventas con descuentos

Este programa calcula el precio final de una venta aplicando un descuento porcentual.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtProducto`.
  - `TextBox`: **Name:** `txtPrecio`.
  - `TextBox`: **Name:** `txtDescuento`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Venta".
  - `ListBox`: **Name:** `lstVentas`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    Dim precio As Double = Val(txtPrecio.Text)
    Dim descuento As Double = Val(txtDescuento.Text)
    Dim total As Double = precio - (precio * descuento / 100)
    lstVentas.Items.Add(txtProducto.Text & " - Precio: " & precio & " -
Descuento: " & descuento & "% - Total: " & total)
    txtProducto.Clear()
    txtPrecio.Clear()
    txtDescuento.Clear()
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnAgregar`, los valores numéricos de `txtPrecio` y `txtDescuento` se obtienen.
  - El `total` se calcula aplicando el descuento: `precio - (precio * descuento / 100)`.
  - Una cadena que incluye el producto, precio original, descuento y el total final se añade a `lstVentas`.
  - Los `TextBox` `txtProducto`, `txtPrecio` y `txtDescuento` se limpian.

Programa 261. Sistema escolar básico

Este programa implementa un sistema escolar básico que permite registrar alumnos y materias en una lista.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtAlumno`.
  - `TextBox`: **Name:** `txtMateria`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Registro".
  - `ListBox`: **Name:** `lstRegistros`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstRegistros.Items.Add("Alumno: " & txtAlumno.Text & " - Materia: " &
txtMateria.Text)
    txtAlumno.Clear()
    txtMateria.Clear()
End Sub
```

- **Explicación:**
  - Cuando se hace clic en `btnAgregar`, el evento `btnAgregar_Click` se activa.
  - El texto de `txtAlumno` y `txtMateria` se concatena en una cadena "Alumno: X - Materia: Y", la cual se añade como un nuevo elemento a `lstRegistros`.
  - Los `TextBox` `txtAlumno` y `txtMateria` se limpian.

Programa 262. Sistema escolar con base de datos

Este programa utiliza un `DataTable` como simulación de una base de datos para almacenar registros de alumnos y materias en un `DataGridView`.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtAlumno`.
  - `TextBox`: **Name:** `txtMateria`.
  - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Registro".
  - `DataGridView`: **Name:** `dgvRegistros`.
- **Código VB.NET:**

```
Dim dt As New DataTable()
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
    dt.Columns.Add("Alumno")
    dt.Columns.Add("Materia")
    dgvRegistros.DataSource = dt
End Sub
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    dt.Rows.Add(txtAlumno.Text, txtMateria.Text)
    txtAlumno.Clear()
    txtMateria.Clear()
End Sub
```

- **Explicación:**
  - Un `DataTable` llamado `dt` se declara a nivel de clase.
  - En `Form1_Load`, se añaden dos columnas al `DataTable`: "Alumno" y "Materia". Luego, `dgvRegistros.DataSource` se enlaza a `dt`, lo que permite que el `DataGridView` muestre los datos del `DataTable`.
  - Cuando se hace clic en `btnAgregar`, una nueva fila con el texto de `txtAlumno` y `txtMateria` se añade a `dt.Rows`. Los `TextBox` se limpian.

Programa 263. Sistema escolar con registro de alumnos

Este programa permite al usuario registrar nuevos alumnos con su nombre y edad en una lista.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtNombreAlumno`.
  - `TextBox`: **Name:** `txtEdad`.
  - `Button`: **Name:** `btnAgregarAlumno`, **Text:** "Agregar Alumno".
  - `ListBox`: **Name:** `lstAlumnos`.
- **Código VB.NET:**

```
Private Sub btnAgregarAlumno_Click(sender As Object, e As EventArgs)
Handles btnAgregarAlumno.Click
    lstAlumnos.Items.Add("Nombre: " & txtNombreAlumno.Text & " - Edad: " &
txtEdad.Text)
    txtNombreAlumno.Clear()
    txtEdad.Clear()
End Sub
```

- **Explicación:**
  - Cuando el usuario hace clic en `btnAgregarAlumno`, el evento `btnAgregarAlumno_Click` se activa.
  - El nombre y la edad del alumno se concatenan en una cadena "Nombre: X - Edad: Y" y se añaden como un nuevo elemento a `lstAlumnos`.
  - Los `TextBox` `txtNombreAlumno` y `txtEdad` se limpian.

Programa 264. Sistema escolar con registro de profesores

Este programa permite registrar nuevos profesores con su nombre y la materia que imparten en una lista.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtNombreProfesor`.
  - `TextBox`: **Name:** `txtMateria`.
  - `Button`: **Name:** `btnAgregarProfesor`, **Text:** "Agregar Profesor".
  - `ListBox`: **Name:** `lstProfesores`.
- **Código VB.NET:**

```
Private Sub btnAgregarProfesor_Click(sender As Object, e As EventArgs)
Handles btnAgregarProfesor.Click
    lstProfesores.Items.Add("Profesor: " & txtNombreProfesor.Text & " -
Materia: " & txtMateria.Text)
    txtNombreProfesor.Clear()
    txtMateria.Clear()
End Sub
```

- **Explicación:**
  - Cuando el usuario hace clic en `btnAgregarProfesor`, el evento `btnAgregarProfesor_Click` se activa.
  - El nombre del profesor y la materia se concatenan en una cadena "Profesor: X - Materia: Y" y se añaden como un nuevo elemento a `lstProfesores`.
  - Los `TextBox` `txtNombreProfesor` y `txtMateria` se limpian.

Programa 265. Sistema escolar con registro de materias

Este programa permite al usuario agregar nuevas materias a una lista.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtMateria`.
  - `Button`: **Name:** `btnAgregarMateria`, **Text:** "Agregar Materia".
  - `ListBox`: **Name:** `lstMaterias`.
- **Código VB.NET:**

```
Private Sub btnAgregarMateria_Click(sender As Object, e As EventArgs)
Handles btnAgregarMateria.Click
    lstMaterias.Items.Add(txtMateria.Text)
    txtMateria.Clear()
End Sub
```

- **Explicación:**
  - Cuando el usuario hace clic en `btnAgregarMateria`, el texto del `txtMateria` se añade como un nuevo elemento a `lstMaterias`.
  - El `txtMateria` se limpia después de agregar la materia.

Programa 266. Sistema escolar con registro de calificaciones

Este programa permite registrar calificaciones de alumnos para una materia específica.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtAlumno`.
  - `TextBox`: **Name:** `txtMateria`.
  - `TextBox`: **Name:** `txtCalificacion`.
  - `Button`: **Name:** `btnAgregarCalificacion`, **Text:** "Agregar Calificación".
  - `ListBox`: **Name:** `lstCalificaciones`.
- **Código VB.NET:**

```
Private Sub btnAgregarCalificacion_Click(sender As Object, e As EventArgs)
Handles btnAgregarCalificacion.Click
    lstCalificaciones.Items.Add("Alumno: " & txtAlumno.Text & " - Materia:
" & txtMateria.Text & " - Nota: " & txtCalificacion.Text)
    txtAlumno.Clear()
    txtMateria.Clear()
    txtCalificacion.Clear()
End Sub
```

- **Explicación:**
  - Cuando el usuario hace clic en `btnAgregarCalificacion`, el nombre del alumno, la materia y la calificación se concatenan en una cadena y se añaden como un nuevo elemento a `lstCalificaciones`.
  - Los `TextBox` correspondientes se limpian.

Programa 267. Sistema escolar con reportes

Este programa genera un reporte simple de las calificaciones, mostrando todos los elementos listados en el `ListBox` en un `MessageBox`.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnReporte`, **Text:** "Generar Reporte".
  - `ListBox`: **Name:** `lstCalificaciones`.
- **Código VB.NET:**

```
Private Sub btnReporte_Click(sender As Object, e As EventArgs) Handles
btnReporte.Click
    Dim reporte As String = "Reporte de Calificaciones:" &
Environment.NewLine
    For Each item In lstCalificaciones.Items
        reporte &= item.ToString() & Environment.NewLine
    Next
    MessageBox.Show(reporte)
End Sub
```

- **Explicación:**
  - Cuando se hace clic en `btnReporte`, el evento `btnReporte_Click` se ejecuta.
  - Se inicializa una cadena `reporte` con el título "Reporte de Calificaciones:" seguido de un salto de línea.
  - Un bucle `For Each` itera sobre cada `item` en `lstCalificaciones.Items`, concatenando cada elemento (convertido a cadena) con un salto de línea a la cadena `reporte`.
  - Finalmente, el `reporte` completo se muestra en un `MessageBox`.

Programa 268. Sistema escolar con gráficos

Este programa muestra un gráfico de columnas en un control `Chart` para visualizar las calificaciones.

- **Controles necesarios en el formulario:**
  - `Chart`: **Name:** `chartCalificaciones`, **ChartType:** `Column`.
  - `Button`: **Name:** `btnGraficar`, **Text:** "Mostrar Gráfico".
  - `ListBox`: **Name:** `lstCalificaciones`.
- **Requisito adicional:**
  - Asegurarse de que el control `Chart` esté disponible en el `Toolbox` (puede requerir añadir la referencia `System.Windows.Forms.DataVisualization` al proyecto).
- **Código VB.NET:**

```
Private Sub btnGraficar_Click(sender As Object, e As EventArgs) Handles
btnGraficar.Click
    chartCalificaciones.Series.Clear()
    Dim serie As New DataVisualization.Charting.Series("Calificaciones")
    serie.ChartType = DataVisualization.Charting.SeriesChartType.Column
    For Each item In lstCalificaciones.Items
        serie.Points.AddXY(item.ToString(), 1)
    Next
    chartCalificaciones.Series.Add(serie)
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnGraficar`, el evento `btnGraficar_Click` se activa.
  - `chartCalificaciones.Series.Clear()` limpia cualquier serie de datos existente en el gráfico.
  - Se crea una nueva serie llamada "Calificaciones" y su `ChartType` se establece en `Column`.
  - Un bucle `For Each` itera sobre cada `item` en `lstCalificaciones.Items`.
  - Para cada ítem, se añade un punto al gráfico con el nombre del ítem y un valor de 1, representando la existencia de cada calificación.
  - Finalmente, la `serie` se añade a `chartCalificaciones.Series`, lo que hace que el gráfico se muestre.

Programa 269. Sistema escolar con exportación Excel

Este programa conceptual simula la exportación de datos de calificaciones a una hoja de cálculo de Microsoft Excel.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnExportar`, **Text:** "Exportar a Excel".
- **Requisito adicional:**
    - Este programa es conceptual y requeriría agregar una referencia a la **biblioteca de objetos de Microsoft Excel** en el proyecto para una implementación real.
- **Código VB.NET:**

```
Imports Microsoft.Office.Interop.Excel
Private Sub btnExportar_Click(sender As Object, e As EventArgs) Handles
btnExportar.Click
    MessageBox.Show("Calificaciones exportadas a Excel (requiere
referencia a Microsoft Excel)")
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnExportar`, se muestra un `MessageBox` conceptual que indica que las "Calificaciones" han sido "exportadas a Excel" y recuerda la necesidad de la referencia a la librería de Excel.

Programa 270. Sistema escolar con importación Excel

Este programa conceptual simula la importación de datos de calificaciones desde una hoja de cálculo de Microsoft Excel.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnImportar`, **Text:** "Importar desde Excel".
- **Requisito adicional:**
    - Este programa es conceptual y requeriría agregar una referencia a la **biblioteca de objetos de Microsoft Excel** en el proyecto para una implementación real.
- **Código VB.NET:**

```
Imports Microsoft.Office.Interop.Excel
Private Sub btnImportar_Click(sender As Object, e As EventArgs) Handles
btnImportar.Click
    MessageBox.Show("Calificaciones importadas desde Excel (requiere
referencia a Microsoft Excel)")
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnImportar`, se muestra un `MessageBox` conceptual que indica que las "Calificaciones" han sido "importadas desde Excel" y recuerda la necesidad de la referencia a la librería de Excel.

Programa 271. Sistema escolar con múltiples usuarios

Este programa simula el inicio de sesión para diferentes tipos de usuarios (Admin, Profesor, Alumno) seleccionados de un `ComboBox`.

- **Controles necesarios en el formulario:**
    - `ComboBox`: **Name:** `cmbUsuarios`, **Items:** "Admin", "Profesor", "Alumno".
    - `Button`: **Name:** `btnAcceder`, **Text:** "Acceder".
- **Código VB.NET:**

```
Private Sub btnAcceder_Click(sender As Object, e As EventArgs) Handles
btnAcceder.Click
    MessageBox.Show("Usuario: " & cmbUsuarios.SelectedItem & " ha iniciado
sesión")
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnAcceder`, se muestra un `MessageBox` que indica qué tipo de usuario ha "iniciado sesión" basándose en la selección del `cmbUsuarios`. Este es un ejemplo conceptual y no incluye una autenticación real de credenciales.

Programa 272. Sistema escolar con permisos

Este programa demuestra cómo la acción de "eliminar registro" puede estar restringida a ciertos roles, como "Admin", seleccionados de un `ComboBox`.

- **Controles necesarios en el formulario:**
  - `ComboBox`: **Name:** `cmbRoles`, **Items:** "Admin", "Profesor".
  - `Button`: **Name:** `btnEliminarRegistro`, **Text:** "Eliminar Registro".
- **Código VB.NET:**

```
Private Sub btnEliminarRegistro_Click(sender As Object, e As EventArgs)
Handles btnEliminarRegistro.Click
    If cmbRoles.SelectedItem = "Admin" Then
        MessageBox.Show("Registro eliminado")
    Else
        MessageBox.Show("No tiene permisos para eliminar")
    End If
End Sub
```

- **Explicación:**
  - Cuando se hace clic en `btnEliminarRegistro`, el código verifica si el rol seleccionado en `cmbRoles` es "Admin". Si es "Admin", se muestra "Registro eliminado"; de lo contrario, se indica que "No tiene permisos para eliminar".

Programa 273. Sistema escolar con auditoría

Este programa registra la adición de alumnos en un `ListBox` de auditoría, incluyendo la fecha y hora del evento.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtAlumno`.
  - `Button`: **Name:** `btnAgregarAlumno`, **Text:** "Agregar Alumno".
  - `ListBox`: **Name:** `lstAuditoria`.
- **Código VB.NET:**

```
Private Sub btnAgregarAlumno_Click(sender As Object, e As EventArgs)
Handles btnAgregarAlumno.Click
    lstAuditoria.Items.Add(DateTime.Now.ToString() & " - Alumno agregado:
" & txtAlumno.Text)
    txtAlumno.Clear()
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnAgregarAlumno`, se añade una entrada a `lstAuditoria` que incluye la fecha y hora actuales y el nombre del alumno agregado. El `txtAlumno` se limpia después de la operación.

Programa 274. Sistema escolar con conexión remota

Este programa es un ejemplo conceptual que simula la conexión a un servidor remoto para la gestión del sistema escolar.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnConectar`, **Text:** "Conectar a Servidor".
- **Código VB.NET:**

```
Private Sub btnConectar_Click(sender As Object, e As EventArgs) Handles
btnConectar.Click
    MessageBox.Show("Conexión remota establecida (simulada)")
End Sub
```

- **Explicación:**
    - Cuando se hace clic en `btnConectar`, se muestra un `MessageBox` que indica conceptualmente que se ha establecido una "Conexión remota". Se aclara que esto es una simulación y que una implementación real requeriría código de red complejo.

Programa 275. Sistema escolar con backup automático

Este programa conceptual simula la realización de un backup automático de los datos del sistema escolar.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnBackup`, **Text:** "Realizar Backup".
- **Código VB.NET:**

```
Private Sub btnBackup_Click(sender As Object, e As EventArgs) Handles
btnBackup.Click
    MessageBox.Show("Backup automático de datos escolares realizado
correctamente")
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnBackup`, se muestra un `MessageBox` conceptual que confirma que el "Backup automático de datos escolares" ha sido "realizado correctamente". Una implementación real implicaría copiar archivos de base de datos o exportar datos.

Programa 276. Sistema escolar con restauración de datos

Este programa conceptual simula la restauración de datos del sistema escolar a partir de un backup.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnRestaurar`, **Text:** "Restaurar Backup".
- **Código VB.NET:**

```
Private Sub btnRestaurar_Click(sender As Object, e As EventArgs) Handles
btnRestaurar.Click
    MessageBox.Show("Datos escolares restaurados desde backup")
End Sub
```

- **Explicación:**
  - Cuando se hace clic en `btnRestaurar`, se muestra un `MessageBox` que indica conceptualmente que los "Datos escolares" han sido "restaurados desde backup". Una implementación real implicaría reemplazar archivos de base de datos o importar datos de un backup anterior.

Programa 277. Sistema escolar con notificaciones

Este programa muestra una notificación en el área de la bandeja del sistema (system tray) para alertar al usuario sobre un nuevo evento registrado en el sistema escolar.

- **Controles necesarios en el formulario:**
  - `NotifyIcon`: **Name:** `NotifyIcon1`, **Icon:** `icono.ico`.
  - `Button`: **Name:** `btnNotificar`, **Text:** "Notificar Evento".
- **Código VB.NET:**

```
Private Sub btnNotificar_Click(sender As Object, e As EventArgs) Handles
btnNotificar.Click
    NotifyIcon1.Visible = True
    NotifyIcon1.ShowBalloonTip(3000, "Sistema Escolar", "Nuevo evento
registrado", ToolTipIcon.Info)
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnNotificar`, `NotifyIcon1.Visible` se establece en `True` para asegurar que el ícono sea visible en la bandeja del sistema. Luego, `NotifyIcon1.ShowBalloonTip()` muestra una notificación en forma de globo con una duración de 3 segundos, un título "Sistema Escolar", el mensaje "Nuevo evento registrado" y un `ToolTipIcon.Info`.

Programa 278. Sistema escolar con estadísticas

Este programa muestra estadísticas básicas del sistema escolar, como el número total de registros (calificaciones, etc.) en un `MessageBox`.

- **Controles necesarios en el formulario:**
    - `ListBox`: **Name:** `lstCalificaciones`.
    - `Button`: **Name:** `btnEstadisticas`, **Text:** "Mostrar Estadísticas".
- **Código VB.NET:**

```
Private Sub btnEstadisticas_Click(sender As Object, e As EventArgs)
Handles btnEstadisticas.Click
    Dim totalAlumnos As Integer = lstCalificaciones.Items.Count
    MessageBox.Show("Total de registros: " & totalAlumnos)
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnEstadisticas`, se obtiene el número total de elementos en `lstCalificaciones.Items.Count`. Un `MessageBox` muestra este `totalAlumnos` (representando el total de registros en este contexto) como una estadística simple.

Programa 279. Sistema escolar con boletines

Este programa genera un boletín de calificaciones simple, mostrando todos los elementos listados en el `ListBox` en un `MessageBox`.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnGenerarBoletin`, **Text:** "Generar Boletín".
    - `ListBox`: **Name:** `lstCalificaciones`.
- **Código VB.NET:**

```
Private Sub btnGenerarBoletin_Click(sender As Object, e As EventArgs)
Handles btnGenerarBoletin.Click
    Dim boletin As String = "Boletín de Calificaciones:" &
Environment.NewLine
    For Each item In lstCalificaciones.Items
        boletin &= item.ToString() & Environment.NewLine
    Next
    MessageBox.Show(boletin)
End Sub
```

- **Explicación:**
    - Cuando se hace clic en `btnGenerarBoletin`, se inicializa una cadena `boletin` con un título. Un bucle `For Each` itera sobre cada `item` en `lstCalificaciones.Items`, concatenando cada elemento con un salto de línea a la cadena `boletin`. Finalmente, el `boletin` completo se muestra en un `MessageBox`.

Programa 280. Sistema escolar con login de usuarios

Este programa implementa un sistema de login básico para el sistema escolar con un usuario y contraseña predefinidos.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtUsuario`.
  - `TextBox`: **Name:** `txtContraseña`, **PasswordChar:** `*`.
  - `Button`: **Name:** `btnLogin`, **Text:** "Login".
- **Código VB.NET:**

```
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    If txtUsuario.Text = "admin" And txtContraseña.Text = "1234" Then
        MessageBox.Show("Login exitoso")
    Else
        MessageBox.Show("Usuario o contraseña incorrectos")
    End If
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnLogin`, el código compara el texto de `txtUsuario` con "admin" y `txtContraseña` con "1234". Si ambas condiciones son verdaderas, se muestra "Login exitoso"; de lo contrario, se indica "Usuario o contraseña incorrectos".

Programa 281. Sistema médico básico

Este programa implementa un sistema médico simple que permite registrar pacientes y doctores en una lista.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtPaciente`.
    - `TextBox`: **Name:** `txtDoctor`.
    - `Button`: **Name:** `btnAgregar`, **Text:** "Agregar Registro".
    - `ListBox`: **Name:** `lstRegistros`.
- **Código VB.NET:**

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstRegistros.Items.Add("Paciente: " & txtPaciente.Text & " - Doctor: "
& txtDoctor.Text)
    txtPaciente.Clear()
    txtDoctor.Clear()
End Sub
```

- **Explicación:**
    - Cuando se hace clic en `btnAgregar`, el texto de `txtPaciente` y `txtDoctor` se concatena en una cadena que incluye el nombre del paciente y el doctor, la cual se añade como un nuevo elemento a `lstRegistros`. Los `TextBox` se limpian después de la operación.

Programa 282. Sistema médico con registro de pacientes

Este programa permite al usuario registrar nuevos pacientes con su nombre y edad en una lista.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtNombrePaciente`.
    - `TextBox`: **Name:** `txtEdad`.
    - `Button`: **Name:** `btnAgregarPaciente`, **Text:** "Agregar Paciente".
    - `ListBox`: **Name:** `lstPacientes`.
- **Código VB.NET:**

```
Private Sub btnAgregarPaciente_Click(sender As Object, e As EventArgs)
Handles btnAgregarPaciente.Click
    lstPacientes.Items.Add("Nombre: " & txtNombrePaciente.Text & " - Edad:
" & txtEdad.Text)
    txtNombrePaciente.Clear()
    txtEdad.Clear()
End Sub
```

- **Explicación:**
    - Cuando el usuario hace clic en `btnAgregarPaciente`, el nombre y la edad del paciente se concatenan en una cadena y se añaden como un nuevo elemento a `lstPacientes`. Los `TextBox` se limpian.

Programa 283. Sistema médico con registro de doctores

Este programa permite registrar nuevos doctores con su nombre y la especialidad que tienen en una lista.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtNombreDoctor`.
    - `TextBox`: **Name:** `txtEspecialidad`.
    - `Button`: **Name:** `btnAgregarDoctor`, **Text:** "Agregar Doctor".
    - `ListBox`: **Name:** `lstDoctores`.
- **Código VB.NET:**

```
Private Sub btnAgregarDoctor_Click(sender As Object, e As EventArgs)
Handles btnAgregarDoctor.Click
    lstDoctores.Items.Add("Doctor: " & txtNombreDoctor.Text & " -
Especialidad: " & txtEspecialidad.Text)
    txtNombreDoctor.Clear()
    txtEspecialidad.Clear()
End Sub
```

- **Explicación:**
    - Cuando el usuario hace clic en `btnAgregarDoctor`, el nombre del doctor y la especialidad se concatenan en una cadena y se añaden como un nuevo elemento a `lstDoctores`. Los `TextBox` se limpian.

Programa 284. Sistema médico con registro de citas

Este programa permite registrar citas para pacientes con doctores en una fecha específica utilizando un `DateTimePicker`.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtPaciente`.
    - `TextBox`: **Name:** `txtDoctor`.
    - `DateTimePicker`: **Name:** `dtpFecha`.
    - `Button`: **Name:** `btnAgregarCita`, **Text:** "Agregar Cita".
    - `ListBox`: **Name:** `lstCitas`.
- **Código VB.NET:**

```
Private Sub btnAgregarCita_Click(sender As Object, e As EventArgs) Handles
btnAgregarCita.Click
    lstCitas.Items.Add("Paciente: " & txtPaciente.Text & " - Doctor: " &
txtDoctor.Text & " - Fecha: " & dtpFecha.Value.ToShortDateString())
    txtPaciente.Clear()
    txtDoctor.Clear()
End Sub
```

- **Explicación:**
    - Cuando el usuario hace clic en `btnAgregarCita`, el nombre del paciente, el doctor y la fecha seleccionada en `dtpFecha` se combinan en una cadena y se añaden a `lstCitas`. Los `TextBox` se limpian.

Programa 285. Sistema médico con historial clínico

Este programa permite agregar registros al historial clínico de un paciente, incluyendo un diagnóstico, en una lista.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtPaciente`.
    - `TextBox`: **Name:** `txtDiagnostico`.
    - `Button`: **Name:** `btnAgregarHistorial`, **Text:** "Agregar Historial".
    - `ListBox`: **Name:** `lstHistorial`.
- **Código VB.NET:**

```
Private Sub btnAgregarHistorial_Click(sender As Object, e As EventArgs)
Handles btnAgregarHistorial.Click
    lstHistorial.Items.Add("Paciente: " & txtPaciente.Text & " -
Diagnóstico: " & txtDiagnostico.Text)
    txtPaciente.Clear()
    txtDiagnostico.Clear()
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnAgregarHistorial`, el nombre del paciente y el diagnóstico se concatenan en una cadena y se añaden como un nuevo elemento a `lstHistorial`. Los `TextBox` se limpian.

Programa 286. Sistema médico con facturación

Este programa permite generar una factura para un paciente, mostrando el monto total, en una lista.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtPaciente`.
    - `TextBox`: **Name:** `txtMonto`.
    - `Button`: **Name:** `btnFacturar`, **Text:** "Generar Factura".
    - `ListBox`: **Name:** `lstFacturas`.
- **Código VB.NET:**

```
Private Sub btnFacturar_Click(sender As Object, e As EventArgs) Handles
btnFacturar.Click
    lstFacturas.Items.Add("Paciente: " & txtPaciente.Text & " - Monto: $"
& txtMonto.Text)
    txtPaciente.Clear()
    txtMonto.Clear()
End Sub
```

- **Explicación:**
    - Cuando se hace clic en `btnFacturar`, el nombre del paciente y el monto se combinan en una cadena que incluye el símbolo "$", y se añade como un nuevo elemento a `lstFacturas`. Los `TextBox` se limpian.

Programa 287. Sistema médico con reportes

Este programa genera un reporte simple del historial clínico, mostrando todos los elementos listados en el `ListBox` en un `MessageBox`.

- **Controles necesarios en el formulario:**
  - `Button`: **Name:** `btnReporte`, **Text:** "Generar Reporte".
  - `ListBox`: **Name:** `lstHistorial`.
- **Código VB.NET:**

```
Private Sub btnReporte_Click(sender As Object, e As EventArgs) Handles
btnReporte.Click
    Dim reporte As String = "Reporte de Historial Clínico:" &
Environment.NewLine
    For Each item In lstHistorial.Items
        reporte &= item.ToString() & Environment.NewLine
    Next
    MessageBox.Show(reporte)
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnReporte`, se inicializa una cadena `reporte` con un título. Un bucle `For Each` itera sobre cada `item` en `lstHistorial.Items`, concatenando cada elemento (convertido a cadena) con un salto de línea a la cadena `reporte`. Finalmente, el `reporte` completo se muestra en un `MessageBox`.

Programa 288. Sistema médico con gráficos

Este programa muestra un gráfico de columnas en un control `Chart` para visualizar las citas.

- **Controles necesarios en el formulario:**
  - `Chart`: **Name:** `chartCitas`, **ChartType:** `Column`.
  - `Button`: **Name:** `btnGraficar`, **Text:** "Mostrar Gráfico".
  - `ListBox`: **Name:** `lstCitas`.
- **Requisito adicional:**
  - Asegurarse de que el control `Chart` esté disponible en el `Toolbox` (puede requerir añadir la referencia `System.Windows.Forms.DataVisualization` al proyecto).
- **Código VB.NET:**

```
Private Sub btnGraficar_Click(sender As Object, e As EventArgs) Handles
btnGraficar.Click
    chartCitas.Series.Clear()
    Dim serie As New DataVisualization.Charting.Series("Citas")
    serie.ChartType = DataVisualization.Charting.SeriesChartType.Column
    For Each item In lstCitas.Items
        serie.Points.AddXY(item.ToString(), 1)
    Next
    chartCitas.Series.Add(serie)
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnGraficar`, `chartCitas.Series.Clear()` limpia cualquier serie de datos existente. Se crea una nueva serie llamada "Citas" de tipo columna. Un bucle `For Each` itera sobre cada `item` en `lstCitas.Items`, añadiendo un punto al gráfico con el nombre del ítem y un valor de 1, representando la existencia de cada cita. Finalmente, la `serie` se añade al gráfico.

Programa 289. Sistema médico con exportación Excel

Este programa conceptual simula la exportación de datos médicos a una hoja de cálculo de Microsoft Excel.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnExportar`, **Text:** "Exportar a Excel".
- **Requisito adicional:**
    - Este programa es conceptual y requeriría agregar una referencia a la **biblioteca de objetos de Microsoft Excel** en el proyecto para una implementación real.
- **Código VB.NET:**

```
Imports Microsoft.Office.Interop.Excel
Private Sub btnExportar_Click(sender As Object, e As EventArgs) Handles
btnExportar.Click
    MessageBox.Show("Datos exportados a Excel (requiere referencia a
Microsoft Excel)")
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnExportar`, se muestra un `MessageBox` conceptual que indica que los "Datos exportados a Excel" y recuerda la necesidad de la referencia a la librería de Excel.

Programa 290. Sistema médico con importación Excel

Este programa conceptual simula la importación de datos médicos desde una hoja de cálculo de Microsoft Excel.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnImportar`, **Text:** "Importar desde Excel".
- **Requisito adicional:**
    - Este programa es conceptual y requeriría agregar una referencia a la **biblioteca de objetos de Microsoft Excel** en el proyecto para una implementación real.
- **Código VB.NET:**

```
Imports Microsoft.Office.Interop.Excel
Private Sub btnImportar_Click(sender As Object, e As EventArgs) Handles
btnImportar.Click
    MessageBox.Show("Datos importados desde Excel (requiere referencia a
Microsoft Excel)")
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnImportar`, se muestra un `MessageBox` conceptual que indica que los "Datos importados desde Excel" y recuerda la necesidad de la referencia a la librería de Excel.

Programa 291. Sistema médico con múltiples usuarios

Este programa conceptual simula el inicio de sesión para diferentes tipos de usuarios (Admin, Doctor, Recepción) seleccionados de un `ComboBox`.

- **Controles necesarios en el formulario:**
    - `ComboBox`: **Name:** `cmbUsuarios`, **Items:** "Admin", "Doctor", "Recepción".
    - `Button`: **Name:** `btnAcceder`, **Text:** "Acceder".
- **Código VB.NET:**

```
Private Sub btnAcceder_Click(sender As Object, e As EventArgs) Handles
btnAcceder.Click
    MessageBox.Show("Usuario: " & cmbUsuarios.SelectedItem & " ha iniciado
sesión")
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnAcceder`, se muestra un `MessageBox` que indica qué tipo de usuario ha "iniciado sesión" basándose en la selección del `cmbUsuarios`.

Programa 292. Sistema médico con permisos

Este programa demuestra cómo la acción de "eliminar registro" puede estar restringida a ciertos roles, como "Admin" o "Doctor", seleccionados de un `ComboBox`.

- **Controles necesarios en el formulario:**
    - `ComboBox`: **Name:** `cmbRoles`, **Items:** "Admin", "Doctor".
    - `Button`: **Name:** `btnEliminarRegistro`, **Text:** "Eliminar Registro".
- **Código VB.NET:**

```
Private Sub btnEliminarRegistro_Click(sender As Object, e As EventArgs)
Handles btnEliminarRegistro.Click
    If cmbRoles.SelectedItem = "Admin" Then
        MessageBox.Show("Registro eliminado")
    Else
        MessageBox.Show("No tiene permisos para eliminar")
    End If
End Sub
```

- **Explicación:**
    - Cuando se hace clic en `btnEliminarRegistro`, el código verifica si el rol seleccionado en `cmbRoles` es "Admin". Si es "Admin", se muestra "Registro eliminado"; de lo contrario, se indica que "No tiene permisos para eliminar".

Programa 293. Sistema médico con auditoría

Este programa registra la adición de pacientes en un `ListBox` de auditoría, incluyendo la fecha y hora del evento.

- **Controles necesarios en el formulario:**
    - `TextBox`: **Name:** `txtPaciente`.
    - `Button`: **Name:** `btnAgregarPaciente`, **Text:** "Agregar Paciente".
    - `ListBox`: **Name:** `lstAuditoria`.
- **Código VB.NET:**

```
Private Sub btnAgregarPaciente_Click(sender As Object, e As EventArgs)
Handles btnAgregarPaciente.Click
    lstAuditoria.Items.Add(DateTime.Now.ToString() & " - Paciente
agregado: " & txtPaciente.Text)
    txtPaciente.Clear()
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnAgregarPaciente`, se añade una entrada a `lstAuditoria` que incluye la fecha y hora actuales y el nombre del paciente agregado. El `txtPaciente` se limpia después de la operación.

Programa 294. Sistema médico con conexión remota

Este programa es un ejemplo conceptual que simula la conexión a un servidor remoto para la gestión del sistema médico.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnConectar`, **Text:** "Conectar a Servidor".
- **Código VB.NET:**

```
Private Sub btnConectar_Click(sender As Object, e As EventArgs) Handles
btnConectar.Click
    MessageBox.Show("Conexión remota establecida (simulada)")
End Sub
```

- **Explicación:**
    - Cuando se hace clic en `btnConectar`, se muestra un `MessageBox` que indica conceptualmente que se ha establecido una "Conexión remota". Se aclara que esto es una simulación y que una implementación real requeriría código de red complejo.

Programa 295. Sistema médico con backup automático

Este programa conceptual simula la realización de un backup automático de los datos del sistema médico.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnBackup`, **Text:** "Realizar Backup".
- **Código VB.NET:**

```
Private Sub btnBackup_Click(sender As Object, e As EventArgs) Handles
btnBackup.Click
    MessageBox.Show("Backup automático de datos médicos realizado
correctamente")
End Sub
```

- **Explicación:**
    - Al hacer clic en `btnBackup`, se muestra un `MessageBox` conceptual que confirma que el "Backup automático de datos médicos" ha sido "realizado correctamente". Una implementación real implicaría copiar archivos de base de datos o exportar datos.

Programa 296. Sistema médico con restauración de datos

Este programa conceptual simula la restauración de datos del sistema médico a partir de un backup.

- **Controles necesarios en el formulario:**
    - `Button`: **Name:** `btnRestaurar`, **Text:** "Restaurar Backup".
- **Código VB.NET:**

```
Private Sub btnRestaurar_Click(sender As Object, e As EventArgs) Handles
btnRestaurar.Click
    MessageBox.Show("Datos médicos restaurados desde backup")
End Sub
```

- **Explicación:**
    - Cuando se hace clic en `btnRestaurar`, se muestra un `MessageBox` que indica conceptualmente que los "Datos médicos" han sido "restaurados desde backup". Una implementación real implicaría reemplazar archivos de base de datos o importar datos de un backup anterior.

Programa 297. Sistema médico con notificaciones

Este programa muestra una notificación en el área de la bandeja del sistema (system tray) para alertar al usuario sobre un nuevo evento registrado en el sistema médico.

- **Controles necesarios en el formulario:**
  - `NotifyIcon`: **Name:** `NotifyIcon1`, **Icon:** `icono.ico`.
  - `Button`: **Name:** `btnNotificar`, **Text:** "Notificar Evento".
- **Código VB.NET:**

```
Private Sub btnNotificar_Click(sender As Object, e As EventArgs) Handles
btnNotificar.Click
    NotifyIcon1.Visible = True
    NotifyIcon1.ShowBalloonTip(3000, "Sistema Médico", "Nuevo evento
registrado", ToolTipIcon.Info)
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnNotificar`, `NotifyIcon1.Visible` se establece en `True` para asegurar que el ícono sea visible en la bandeja del sistema. Luego, `NotifyIcon1.ShowBalloonTip()` muestra una notificación en forma de globo con una duración de 3 segundos, un título "Sistema Médico", el mensaje "Nuevo evento registrado" y un `ToolTipIcon.Info`.

Programa 298. Sistema médico con recordatorios de citas

Este programa permite al usuario establecer un recordatorio para una cita, mostrando un mensaje con la fecha programada.

- **Controles necesarios en el formulario:**
  - `DateTimePicker`: **Name:** `dtpCita`.
  - `Button`: **Name:** `btnRecordatorio`, **Text:** "Recordar Cita".
- **Código VB.NET:**

```
Private Sub btnRecordatorio_Click(sender As Object, e As EventArgs)
Handles btnRecordatorio.Click
    MessageBox.Show("Recordatorio: Cita programada para el " &
dtpCita.Value.ToShortDateString())
End Sub
```

- **Explicación:**
  - Cuando el usuario hace clic en `btnRecordatorio`, se muestra un `MessageBox` con un mensaje que incluye la fecha de la cita seleccionada en `dtpCita`, convertida a formato de fecha corta.

Programa 299. Sistema médico con estadísticas

Este programa muestra estadísticas básicas del sistema médico, como el número total de pacientes registrados, en un `MessageBox`.

- **Controles necesarios en el formulario:**
  - `ListBox`: **Name:** `lstPacientes`.
  - `Button`: **Name:** `btnEstadisticas`, **Text:** "Mostrar Estadísticas".
- **Código VB.NET:**

```
Private Sub btnEstadisticas_Click(sender As Object, e As EventArgs)
Handles btnEstadisticas.Click
    Dim totalPacientes As Integer = lstPacientes.Items.Count
    MessageBox.Show("Total de pacientes registrados: " & totalPacientes)
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnEstadisticas`, se obtiene el número total de elementos en `lstPacientes.Items.Count`. Un `MessageBox` muestra este `totalPacientes` como una estadística simple.

Programa 300. Sistema médico con login de usuarios

Este programa implementa un sistema de login básico para el sistema médico con un usuario y contraseña predefinidos.

- **Controles necesarios en el formulario:**
  - `TextBox`: **Name:** `txtUsuario`.
  - `TextBox`: **Name:** `txtContraseña`, **PasswordChar:** `*`.
  - `Button`: **Name:** `btnLogin`, **Text:** "Login".
- **Código VB.NET:**

```
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    If txtUsuario.Text = "admin" And txtContraseña.Text = "1234" Then
        MessageBox.Show("Login exitoso")
    Else
        MessageBox.Show("Usuario o contraseña incorrectos")
    End If
End Sub
```

- **Explicación:**
  - Al hacer clic en `btnLogin`, el código compara el texto de `txtUsuario` con "admin" y `txtContraseña` con "1234". Si ambas condiciones son verdaderas, se muestra "Login exitoso"; de lo contrario, se indica "Usuario o contraseña incorrectos".

Epílogo

Al llegar al final de este libro, hemos recorrido juntos un camino compuesto por **300 ejercicios** que abarcan desde los fundamentos más básicos de la programación en **Visual Basic .NET con formularios** hasta aplicaciones de mayor complejidad. El objetivo ha sido que cada paso, cada ejemplo y cada práctica sirviera como una herramienta clara para avanzar en el aprendizaje, sin perder de vista que programar es un arte tanto como una técnica.

Si algo quisiera dejar como mensaje final es que **la programación no se aprende de un día para otro**: se construye con paciencia, constancia y práctica. Cada error cometido y cada línea de código corregida forman parte esencial del aprendizaje. Este libro no pretende ser un punto final, sino una puerta abierta a nuevos proyectos, ideas y retos que cada lector podrá desarrollar de manera personal.

He escrito estas páginas convencido de que **el conocimiento cobra sentido cuando se comparte**. Ese ha sido mi motor para poner a disposición de todos este trabajo: que lo aprendido no quede solo en mí, sino que llegue a otros y pueda ayudar a crecer.

Y por último, quiero reiterar mi agradecimiento más profundo a **mi mujer, Marisa**, cuyo apoyo y compañía han sido la base de este proyecto. Sin su aliento constante, este sueño de compartir el conocimiento a través de una serie de libros no habría sido posible.

Deseo que lo aprendido aquí te acompañe en tus próximos pasos y que cada ejercicio sea una semilla para nuevos logros. La programación es un camino que no termina: es una invitación continua a imaginar, crear y compartir.